ATLAS Operational Monitoring Data Archival and Visualization

Igor Soloviev, University of California, Irvine
on behalf of ATLAS TDAQ Collaboration

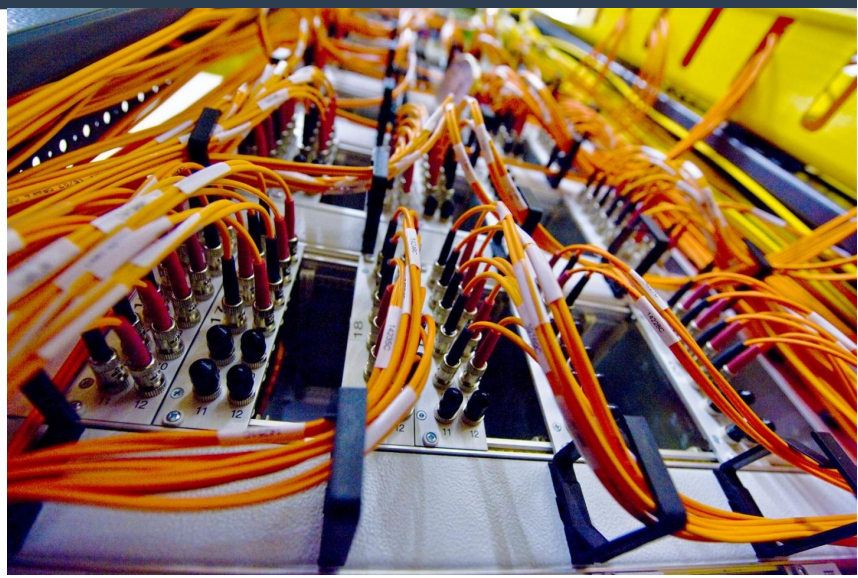
CHEP 2019, Adelaide, Australia
4 November 2019

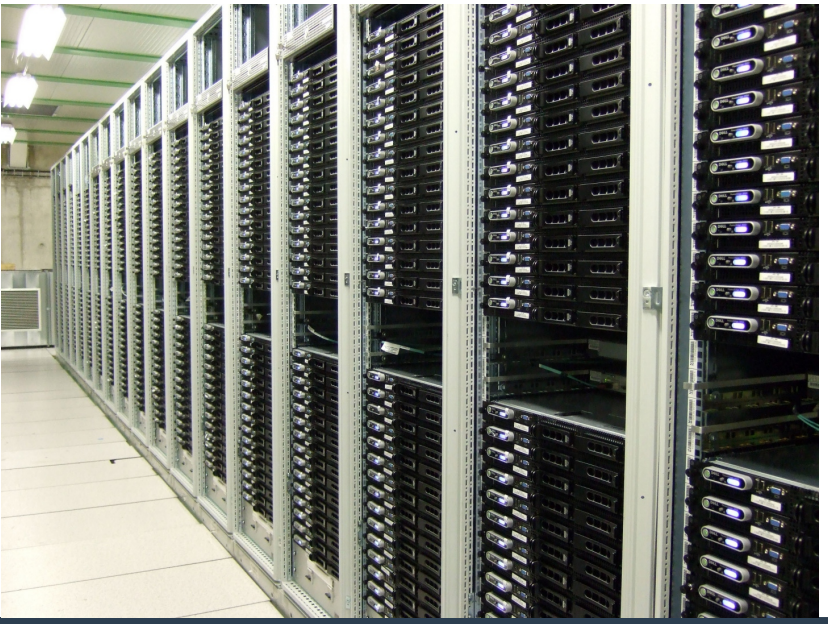
Outline

- **Operational Monitoring in ATLAS**
- **First Prototypes and Implementations**
  - Cassandra and SPLUNK
- **Implementation for Run II**
  - Design
  - Visualization interfaces (Grafana and Beauty)
  - Performance statistics
- **Technology Evaluation**
  - InfluxDB and ClickHouse
- **Status and Plans**

Context

- **ATLAS online:**
  - 100 million electronic channels
  - tens of thousands various applications
  - thousands of computers
  - 1 billion collisions per second,
    up to 1.5 GB/s physics data storage rate

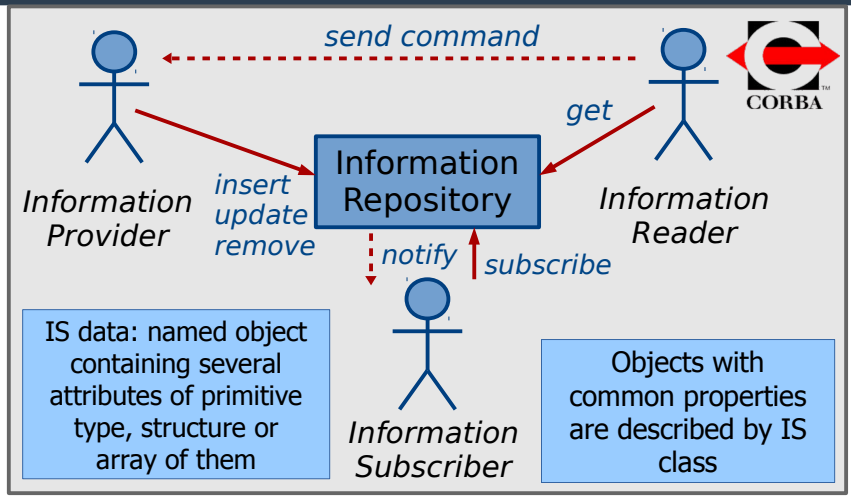




- **The applications produce various operational monitoring data important for successful ATLAS experiment operations**
  - used online for inter-process communication
  - important for experts
  - 200K updates of monitoring data per second

Operational Monitoring Persistence

- **Information Service (IS) has been used in ATLAS since middle of the '90s**
  - CORBA based (reliability, interoperability, efficiency) client-server architecture
  - predefined structure of data (schema), object data model
  - is used by **all** ATLAS online detectors and systems for online operational monitoring defining own schemes

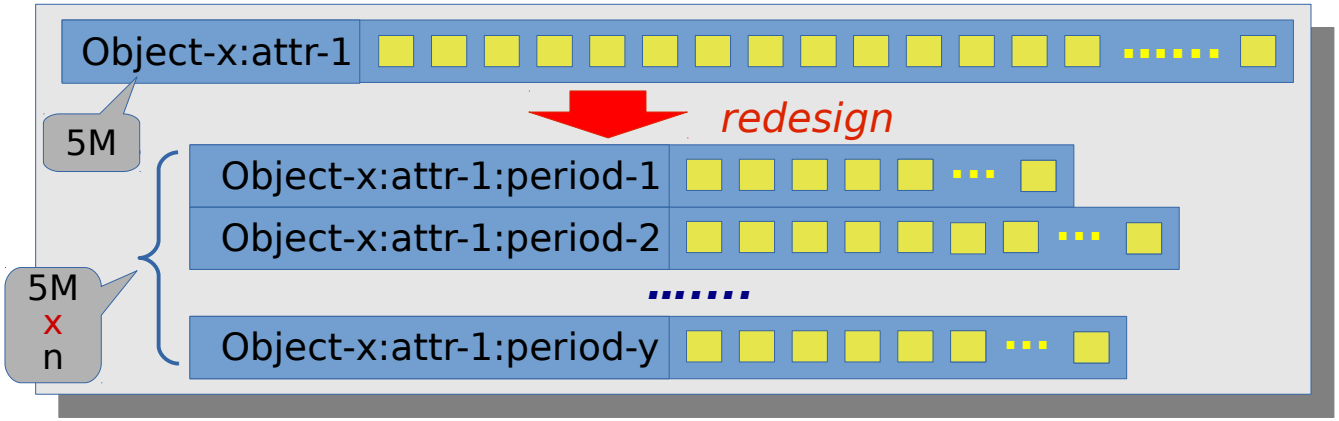


- **Before 2011: no persistency, data are lost after end of data taking**
  - several tools were used by different groups for small subsets of their data, no common solution
- **In 2011 it was suggested to create a common persistence (PBEAST ≡ Persistent Back-End for the ATLAS information System of TDAQ) to aggregate and visualize ATLAS operational data with requirements:**
  - archive a subset of operational monitoring data in generic way
  - long term storage of raw data
  - access via web outside of ATLAS experiment network

About CORBA: https://www.corba.org/

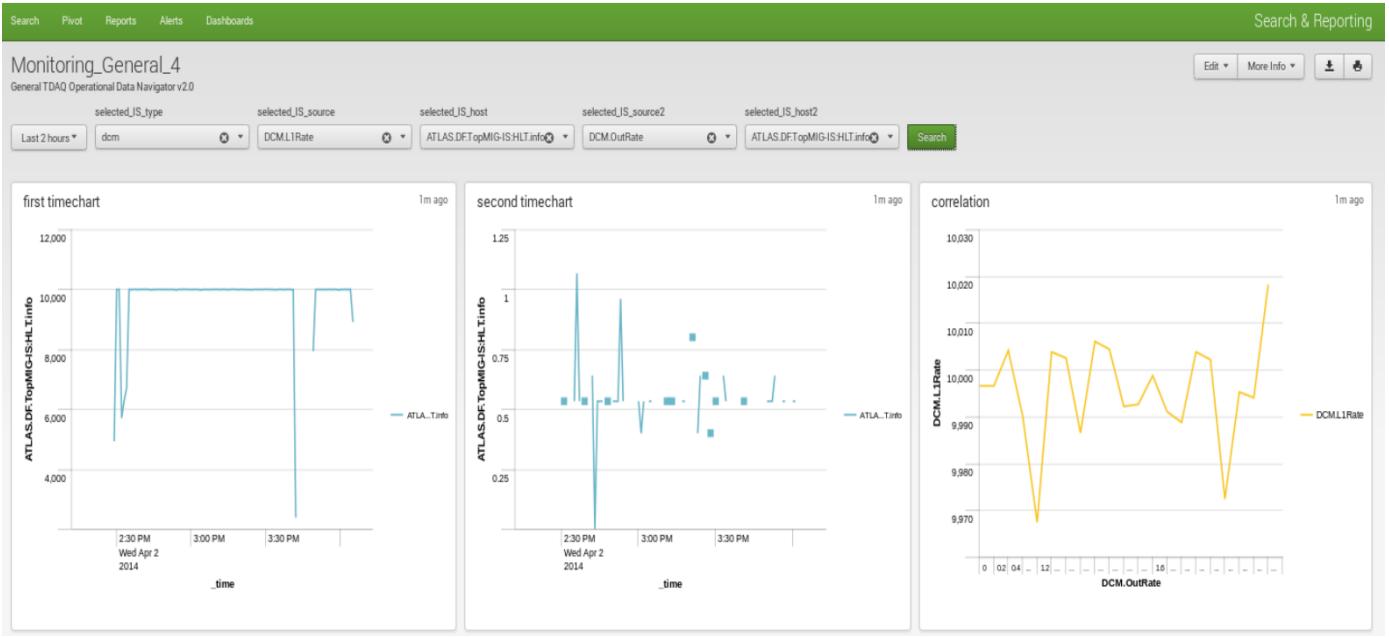
Cassandra Implementation (Run I)

- **Evaluated available technologies in 2012**
  - HBase, MongoDB, Apache Cassandra
- **Chose most promising (Cassandra) – distributed hash table:**
  - freeware, provided cluster solution, linear scalability, fault-tolerance
- **Tried several deployment models and database schemes with real monitoring from ongoing Run-1 during 1.5 years. Problems only become visible during real use. Abandoned by end of 2013:**
  - data append-only architecture; removal of data was over-complicated, space inefficient with various compaction issues, required dedicated maintenance
  - lack of vertical scaling enforced schema redesign (complicated queries) and data smoothing:
    - when size of data row was above 60 MB, discovered serious problems with queries and database compaction
    - not able to store even two months of important raw data acquisition system data (25% of total) on 3x4TB nodes
  - incompatible changes in major versions of Cassandra
  - no arrays and nested types required by the IS data model
  - developed extra tool to export in private JSON format for long term storage and yet another mechanism to access them
  - *see backup slide for more*



SPLUNK Archival and Visualization (Run I)

- **Commercial, cluster license provided by CERN IT for evaluation**
- **Data-to-Everything Platform**
  - distributed non-relational semi-structured time-series database
  - provides general-purpose search, analysis & reporting
- **Evaluated insertion of monitoring data and querying including aggregation, correlation and visualization**



- **Nice all-in-one tool, but:**
  - large disk space overhead (raw text)
  - per day data quota was not sufficient for real use
  - insufficient visualization facilities for scientific data
  - difficult integration with third-party frontend libraries
  - *see backup slide for more*

About SPLUNK: https://www.splunk.com/

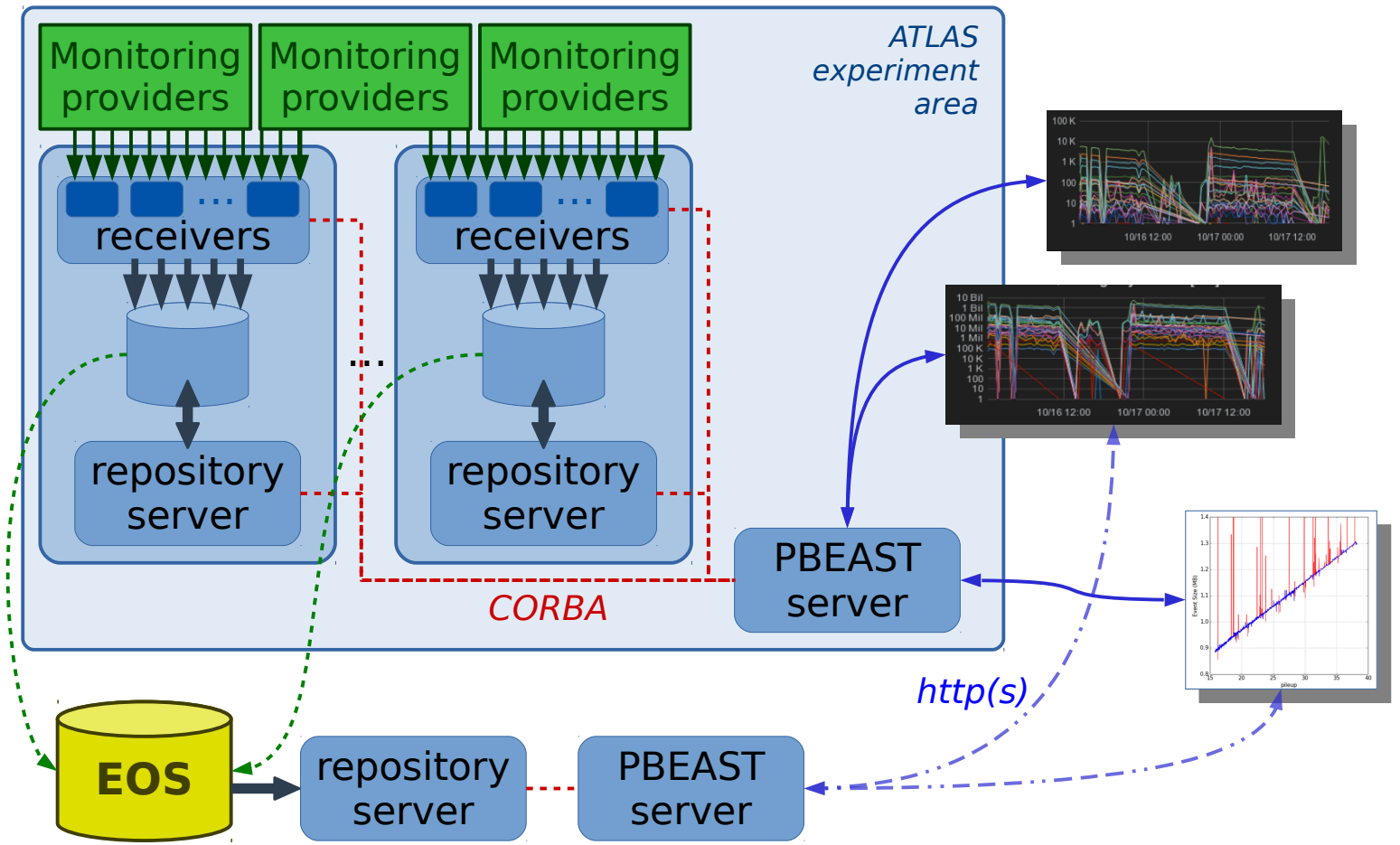
New Implementation (for Run II)

- **We could not afford anymore a risk of yet another failure by start of next run, so in 2014 we decided:**
  - during long shutdown between LHC runs 1 and 2 provide simple and robust solution to store operational monitoring data in files, archive them on long-term storage (EOS) and implement API for data retrieval
- **Solution was based on Google protocol buffers library**
  - efficient binary data serialization including interoperability, compaction and compression
  - private file format using above for random data access
  - the format supports all IS data types and schema evolution
  - monitoring data receiver stores data into files organized by schema and interval buckets
  - many receivers can be configured to work concurrently on several computers to spread the load
  - *see backup slide for more*

About Google Protocol Buffers: https://developers.google.com/protocol-buffers

New PBEAST Service Architecture

- **Implemented a service on top of PBEAST files:**



- **is running on ATLAS experiment site**
- **its internal protocol uses CORBA**
- **PBEAST API (REST, C++, Python, Java) works over http inside experiment area, or https on public network using CERN authentication**
- **PBEAST files are archived on EOS**
- **additional service using EOS is running outside ATLAS experiment area**

About EOS: http://eos-docs.web.cern.ch/eos-docs/

Grafana Data Visualization

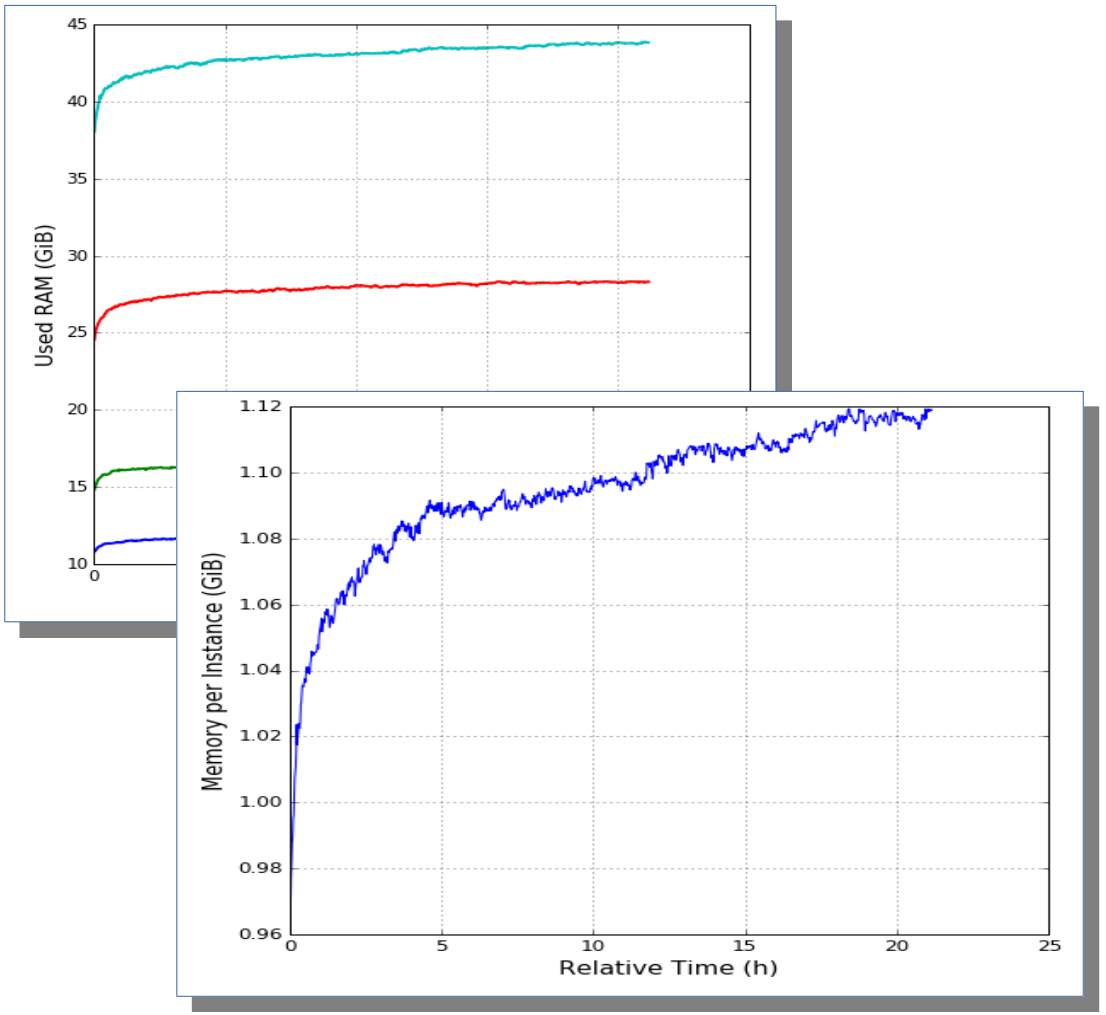
- **Use Grafana since 2014 (after SPLUNK):**
  - open source analytics platform to query and visualize metrics creating and exploring dashboards
  - implemented REST interface from PBEAST service exposing meta information and data to Grafana client
  - implemented Grafana plugin knowing details of PBEAST including support for arrays and downsampled data
- **More than 100 dashboards used by all sub-systems and detectors**
- **Gradually implemented and integrated many useful features:**
  - data averaging instead of sampling
  - aggregation and array functions, aliases, metrics and data filters, annotations
  - new types of plots including scatter plots for correlations, string data representation (discrete plot), tabular views, histograms, pie charts, heat maps
  - LDAP integration for authentication, permissions on folders and dashboards
  - used several Grafana versions from 1.9 to latest 6.x

About Grafana: https://grafana.com/

Grafana Dashboard Example

SWAN Visualization (Beauty)

- **SWAN is a CERN Service for Web based ANalysis**
  - a platform to perform interactive data analysis in the cloud
  - share work and reports with your colleagues using the cloud data storage at CERN (CERNbox)
  - interactive development, immediate results
- **Beauty extends PBEAST for SWAN**
  - recommended to use if Grafana and PBEAST functionality is not enough
- **Any level of algorithms complexity including correlations**
  - Python and C++ API to access data are available
  - interactive plotting including matplotlib and ROOT



SWAN Notebook with Beauty Example

MyBeauty (autosaved)

Control Panel | Logou

File Edit View Insert Cell Kernel Help | Python 2

Code | CellToolbar

reduce the time a lot.

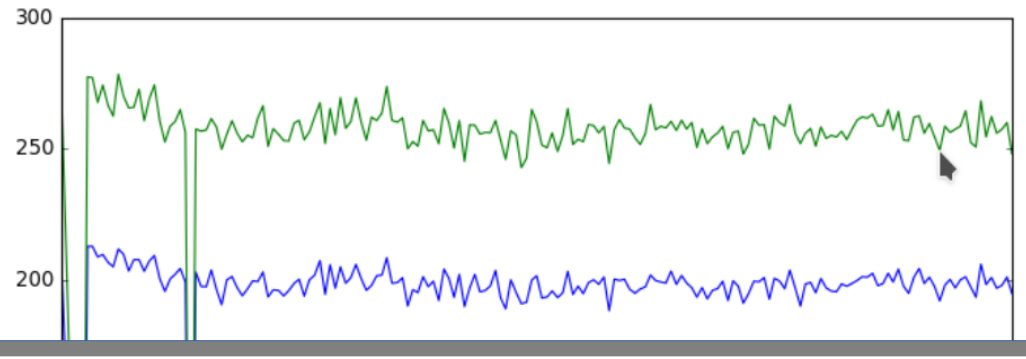
```
In [7]: # Choose your time range
since = dt.datetime(2016, 7, 18, 10, 20, 0)
till  = dt.datetime(2016, 7, 18, 20, 20, 0)

# Retrieve the data for a given interval
rates = TRPTimepoints(since, till,
                      'L1_Rate|HLT_Rate',
                      default_columns={ 'HLT_Rate': 'output', 'L1_Rate': 'TAP'},
                      downsample_interval=600
                     )

# We generate some outp
print "Data has been re

Data has been retrieved
```

Retrieve data

Draw

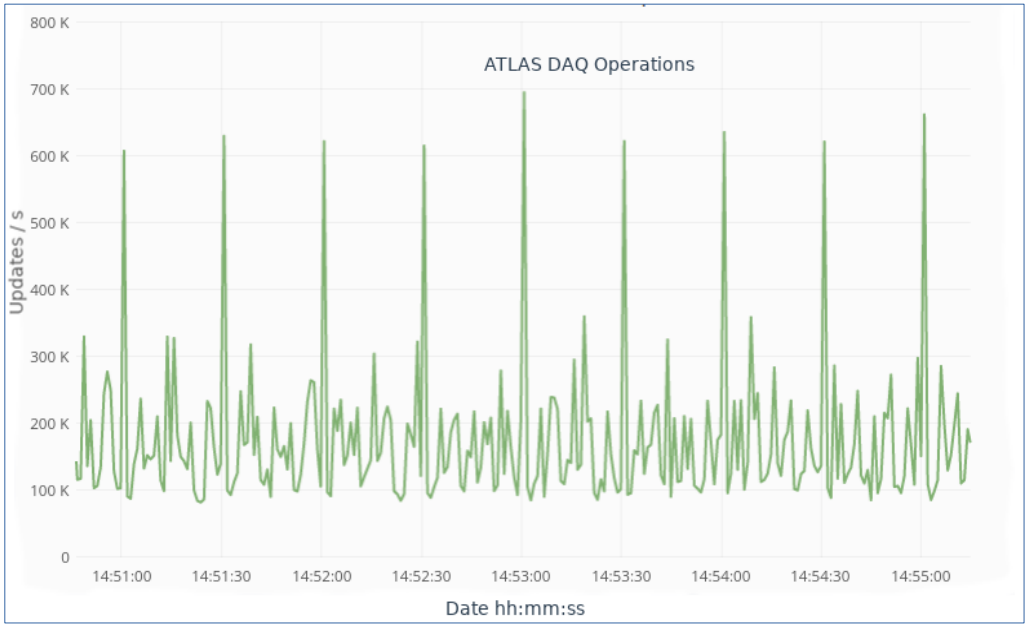
```
In [13]: plt.figure(figsize=(10,8))
plot(run_rates,'HLT_Rate', 'total')
plot(run_rates,'HLT_Rate', 'recording')

# beautify time axis
ax = plt.gca()
ax.xaxis.set_major_locator(mdates.HourLocator())
ax.xaxis.set_major_formatter(mdates.DateFormatter('%H:%M', tz=pytz.timezone('Europe/Zurich'))
```

Figure 3

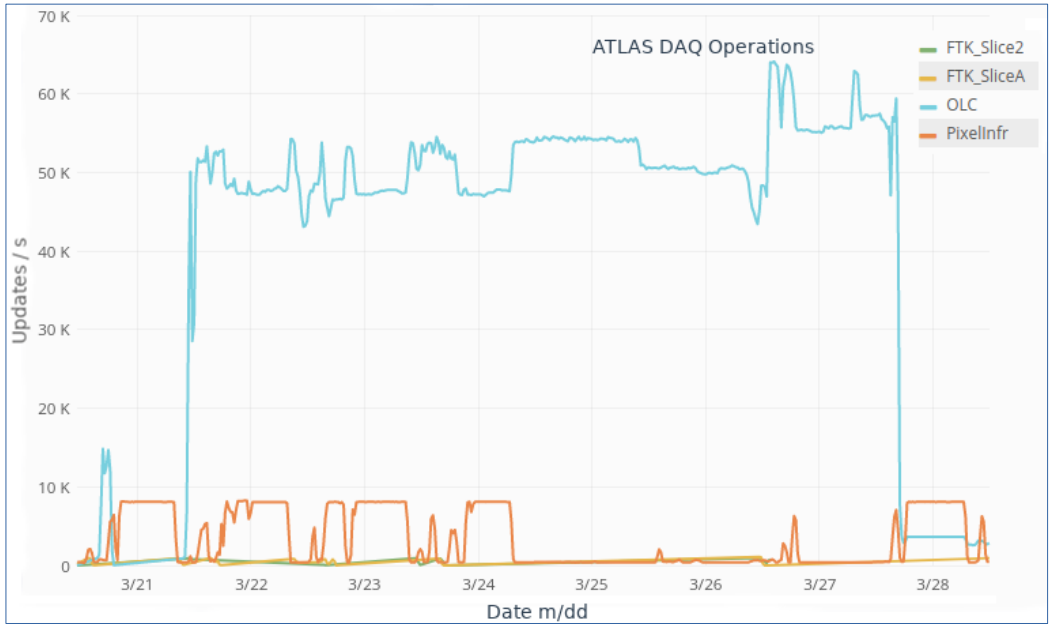


Data Insertion Rates

- **Store all raw operational data from the ATLAS data taking sessions, plus some ancillary data from secondary sources (like the networking system)**
- **ATLAS session has average 180 KHz insertion rate during data taking**
- **Network monitoring has constant rate around 10 KHz**
- **Other sources may contribute significant data as well**



Rate of monitoring data updates in ATLAS data taking session during few minutes of typical data taking period *

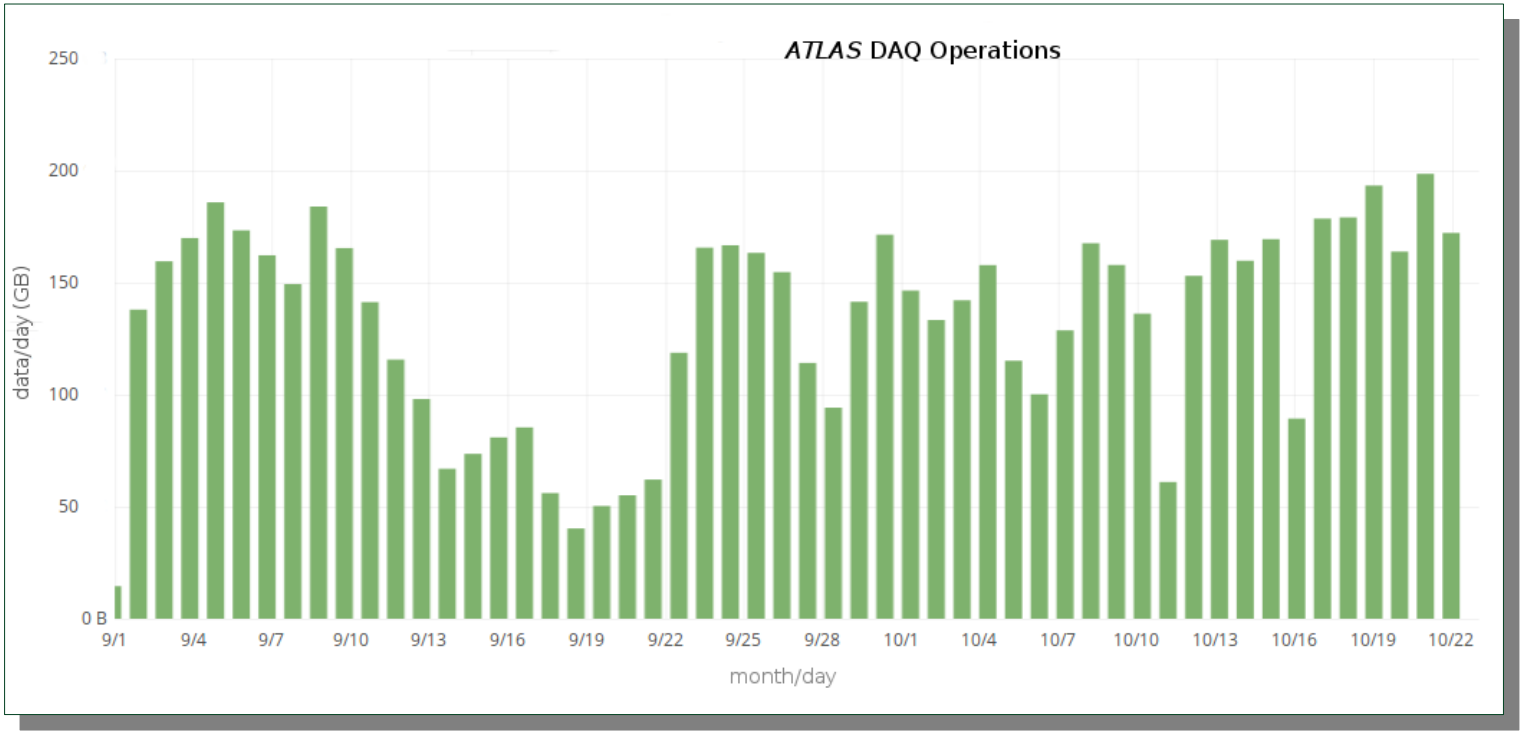


Rate of monitoring data updates in user data taking sessions during second milestone week (M2) in 2018 *

* https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ

Storage Resources

- **Two 2015 32 TB, three 2012 4 TB nodes**
- **1.5 TB / month during data taking periods (compressed)**
- **Store raw data with compaction and compression**
- **Downsample data when required and store on server cache**



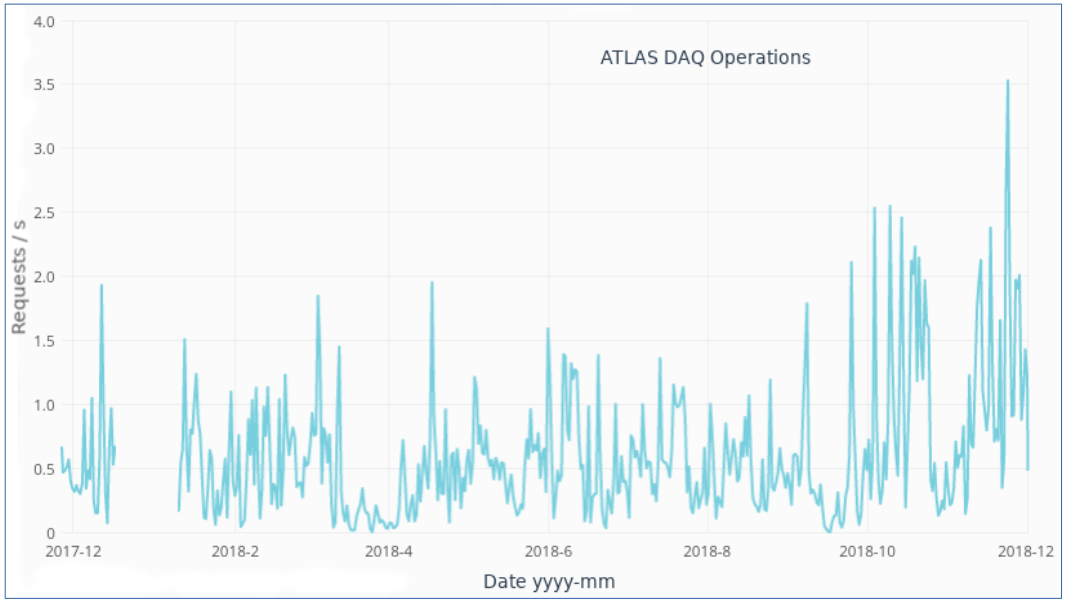
Archived data in PBeast per day, for few days in ATLAS operations in October 2018 *

- **Plan to store raw data for lifetime of ATLAS**
- **New hardware before start of Run 3**

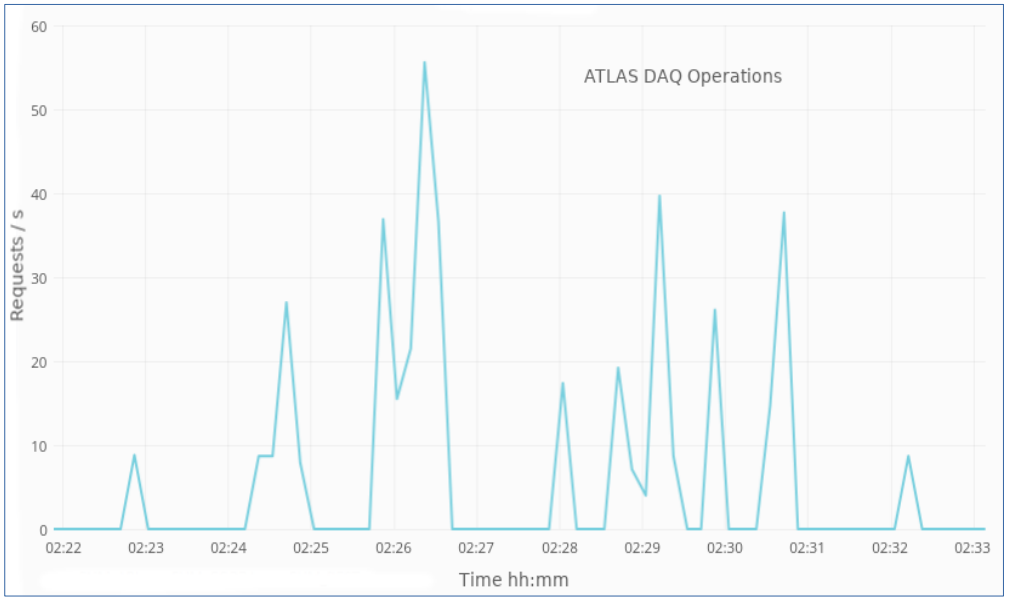
* https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ

Data Access Statistics

- **30-60 Grafana dashboards queries per minute in average**
- **There are short periods with much higher request rates (+50 q/s)**



Average number of PBEAST REST interface requests rate during several months of Run II *



Increase of PBEAST REST interface requests rate during few minutes of a nightly data-taking run on May 18, 2018 *

- **PBEAST is used for ATLAS operations and post-mortem analysis**
- **Provides access to live and historical data inside and outside ATLAS experiment area**

* https://twiki.cern.ch/twiki/bin/view/AtlasPublic/ApprovedPlotsDAQ

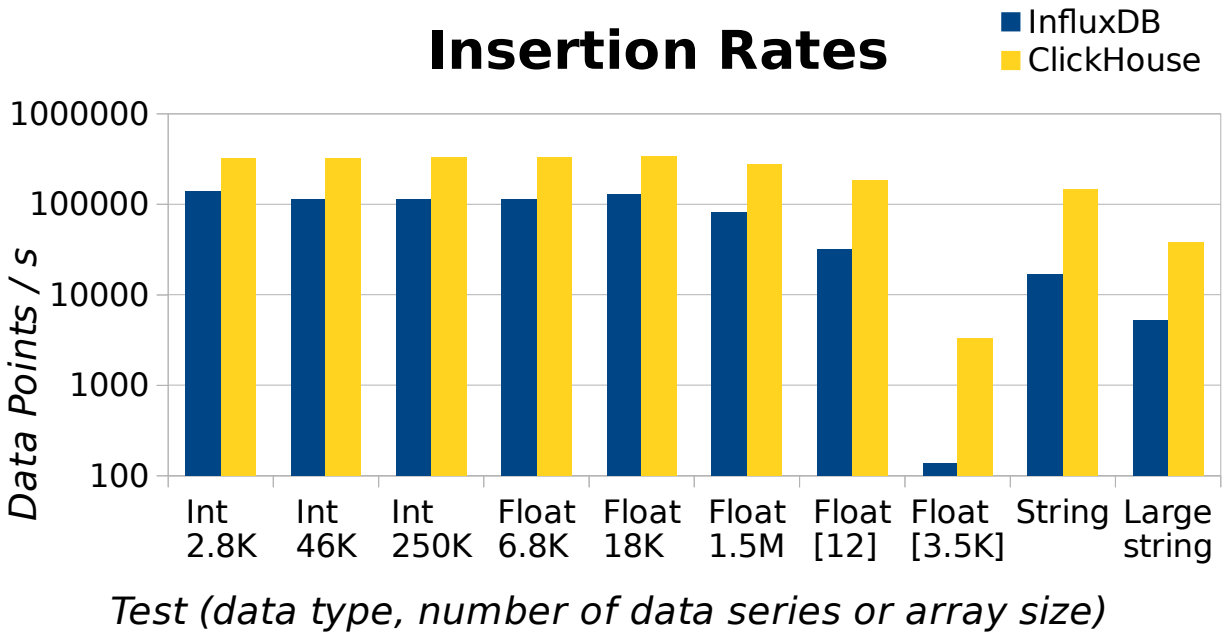
Technology Evaluation

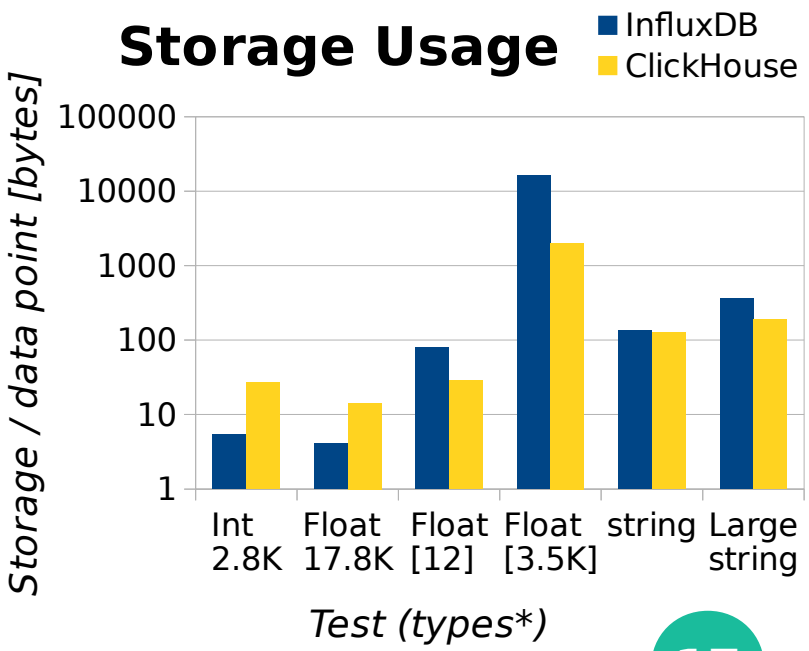
influxdata

- **Perform technology evaluation to see possible candidates for PBEAST service implementation or ideas how to improve it in future**
  - PBEAST **file format** is *rather simple* and *may not be efficient* for certain types of queries
  - no data **fault tolerance** apart of used *RAID* hardware and weekly *replication* on EOS
  - many time-series oriented databases appeared after 2014
- **InfluxDB - open source time series database**
  - optimized for **fast**, **high-availability** storage and retrieval of **time series data**
  - SQL-like language with built-in time-centric functions for querying a data
  - but: **no** support for **arrays**, **cluster** solution is only available in **commercial** enterprise DB
- **ClickHouse (Yandex) - open source column-oriented DBMS**
  - designed for very **fast column-oriented** queries and hardware efficiency
  - feature reach, support of all required data types including arrays, SQL query dialect with a number of built-in analytics capabilities
  - linear **scalability**, distributed reads, **free cluster** and data centers **solution**
  - fault tolerance, highly reliable, automatic synchronization after server downtime
  - but: it is **not time series database**, some functions can be less efficient

First Evaluation Results

- **Designed data models for both technologies allowing schema evolution and arrays support (missing in InfluxDB)**
- **Inserted 4 months of raw PBEAST data and compared insertion rates and storage used per data point by data types**
- **ClickHouse is faster for data insertion and works better with arrays. InfluxDB is optimal for simple numeric values. Both are less efficient than PBEAST, that is highly optimized for online operational monitoring data model.**





Status and Plans

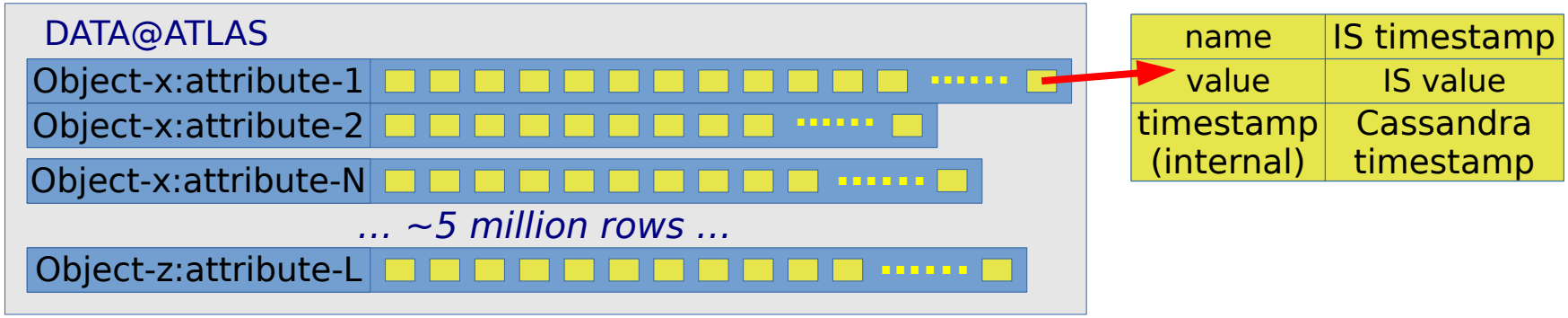
- **PBEAST was successfully used during Run 2 and is planned to be used in Run 3 without major changes [apart of new hardware]**
  - **stability** during development of many required features and improvements during Run-2
  - attractive and **easy-to-use Web tools** (Grafana dashboards, Beauty)
  - **common monitoring technology across experiment** is very important for integration with sub-systems and sub-detectors (monitoring data are archived transparently, just configure dashboard)
- **Implementation on top of ProtoBuf was at least one order of magnitude more efficient in terms of CPU and disk utilization comparing with best available technologies**
  - **one node** accepted the same monitoring data rate, as a **cluster of Cassandra nodes**
  - even modern time-series database technologies have worse results
  - the "**price**" of such PBEAST service implementation was **moderate** from human and hw resources points of view, and, e.g. even **less expensive** than **effort on Cassandra** prototypes (~3 years of FTEs)
- **PBEAST becomes a critical component for detector operations**
  - started as a complimentary monitoring tool, during Run 2, it became **essential for online detector monitoring**, offline analysis of operations data, various operations reports and publications
  - during next run, in addition, it will be used by Trigger community to replace in-house developed GUI and network monitoring
- **The technology evaluation will be continued to make any types of queries efficient, to add easy horizontal scalability, and high availability using cluster solutions**
- **Even more features like trends prediction and alarms are in the plans**

Backup Slides

Cassandra Implementation
SPLUNK Implementation
PBEAST file format

Cassandra implementation details

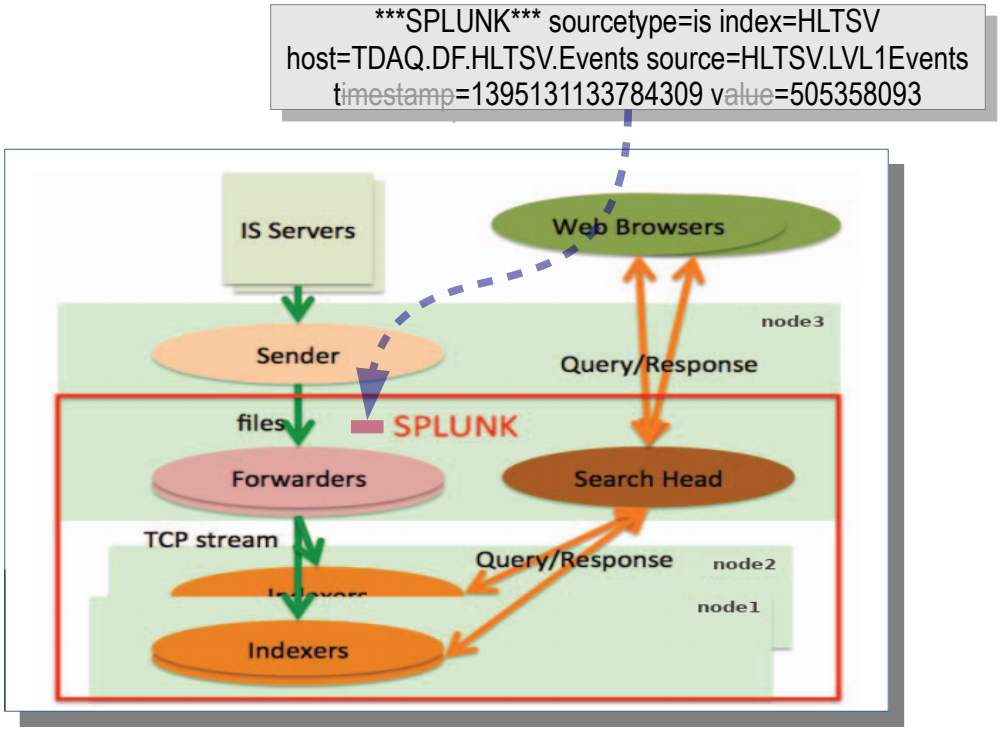
- **Column family (table) stores data by class, object id and attribute rows**
- **Each row contains key:value pairs (key = timestamp, value = attribute value)**

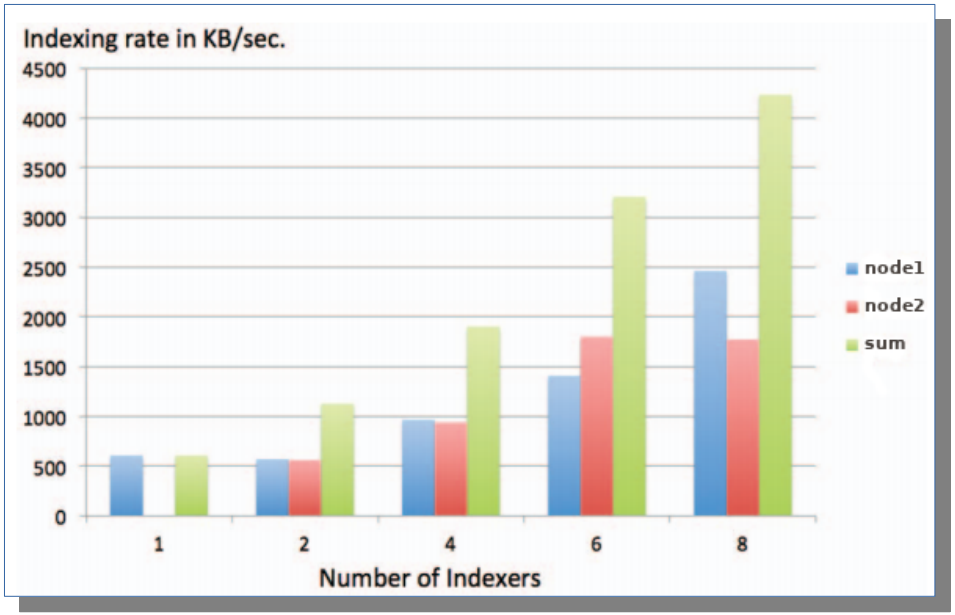


- **Evaluated several models for deployment.**
  Finally installed PBEAST on 3 powerful (year 2012) nodes on ATLAS experiment area:
  - dual 6 cores CPU Xeon X5650 @ 2.67GHz, 24 GB RAM, 4x1TB RAID
- **Use data replication factor 2 (and RAID 0)**
- **Subscribe subset of DAQ IS servers and classes (below 1/4 of total IS information in ATLAS)**
- **6 MB/s writing aggregate performance (0.5 TB / day!) and 18.4K updates per second**
- **Introduce 5% data smoothing (skip “similar” numeric data) to reduce writing to 0.8 MB/s**

SPLUNK implementation details

- **Forwarders, Search Head and Indexers are components of SPLUNK running on different computer nodes for maximum performance**
- **Store monitoring data into a file (stream) in a text format optimised for Splunk**
- **The text format uses ~200 bytes per IS attribute value (compressed by SPLUNK)**

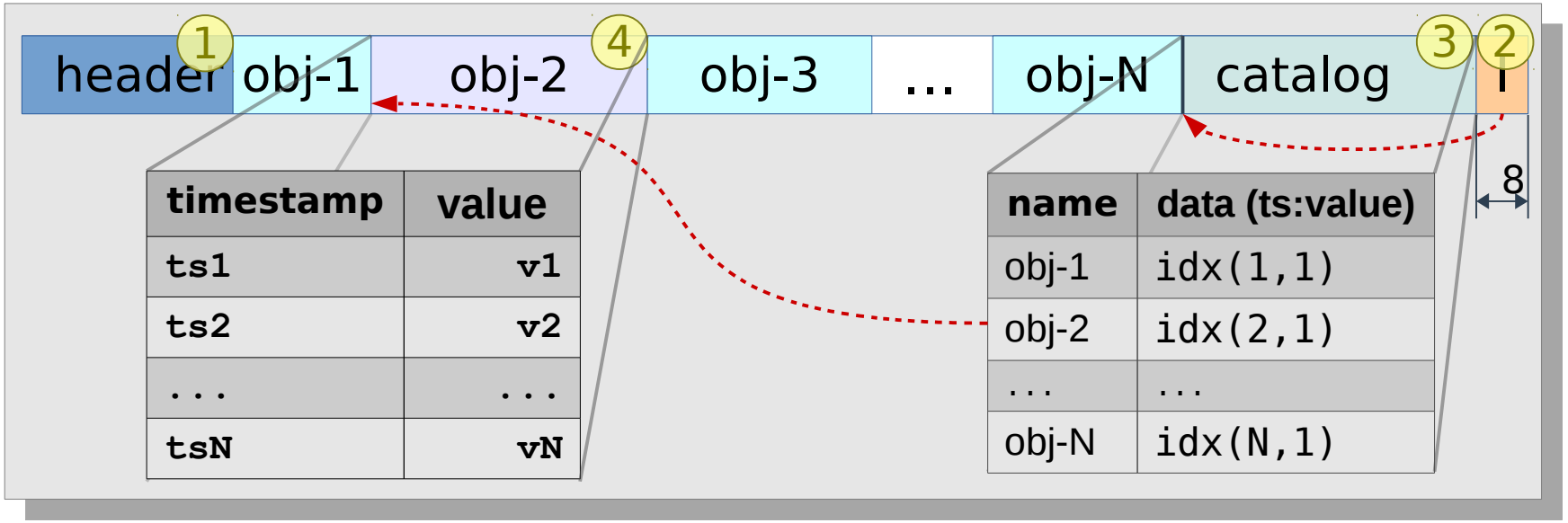




- **Insertion rate scales proportionally to number of indexers and reaches 2 MB/s (~10000 IS updates/s) per single node**
- **Querying performance depends on many indexing parameters and in general case allows to scan about 100,000 values per second and per indexer node**

PBEAST File Format

- **Low-level ProtoBuf primitives using integer variants for compaction (smaller numbers are serialized into smaller number of bytes) and zip compression**
- **Microseconds precision (configurable) for timestamps relative to the file base value**
- **Sequential write at once, no file size limit (2^{64} bytes), zip data per object if necessary. Random read access, efficient on HDDs.**
- **Numeric types, strings, date/time, structures and arrays, end of validity**



- **Header and catalog are read once and stored in cache**
- **Every object is accessible independently**