Design of the Event Metadata System for BM@N and testing the first prototypes

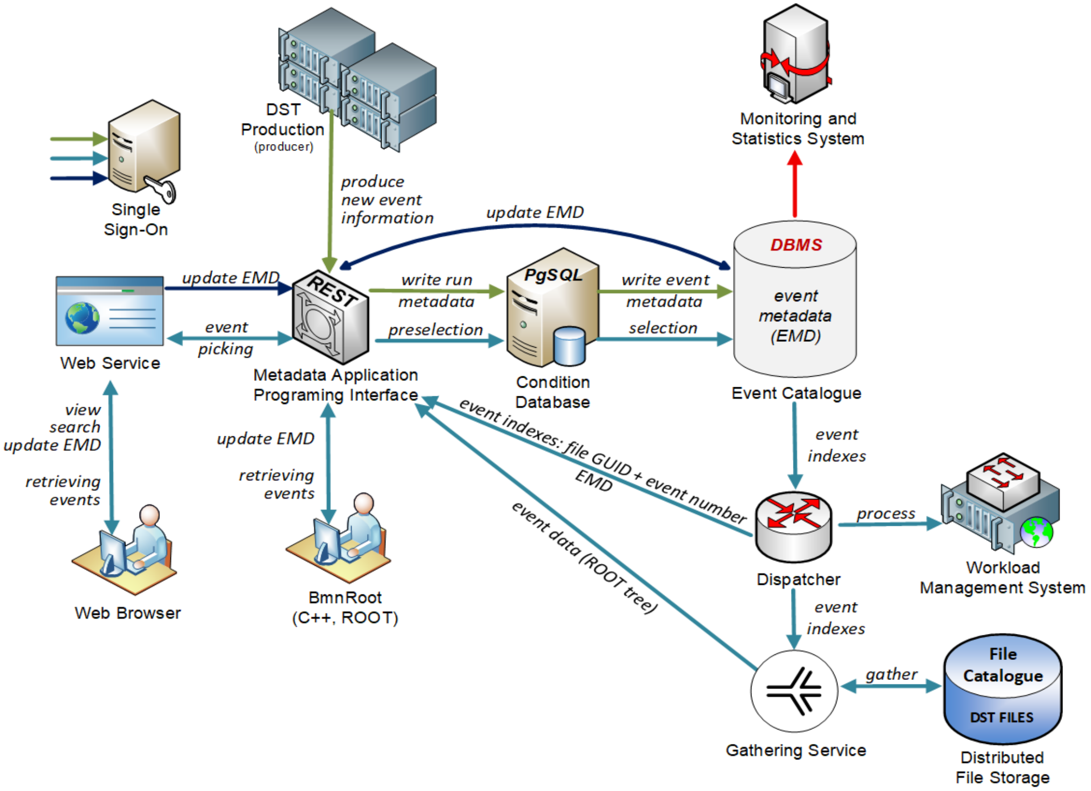
A.Yakovlev

7th Collaboration Meeting of the
BM@N Experiment at the NICA Facility
JINR, Dubna

Event Metadata System

The Event Metadata System based on the Event Database contains physics event metadata, which include specific information on particle collision events, allowing users to quickly search for a required set of events based on various criteria and parameters and select the reconstructed events for further processing and physics analysis.

Event Metadata System
Common architecture



Event Metadata System

Summary event metadata to be stored in the Event Database should be chosen to search and select required events of the experiment by possible requests from collaboration members for various physics analyses.

Reconstructed event information varies in the NICA experiments, therefore a specific set of metadata attributes will be defined at the deployment step of the Event Metadata System using a configuration file.

Event Metadata System

The Event Catalogue will provide summary event data to select necessary events of the NICA experiments by criteria, such as:

- **period number (Run) and run number;**
- **software version;**
- **event time (a start point in time is sufficient to not store a corresponding time interval);**
- **number of primary and all reconstructed tracks;**
- **track number of positively and negatively charged particles from the primary vertex;**
- **primary and secondary particles found;**
- **number of hits by detectors;**
- **total input and output charge in the event;**
- **and others to be used for selecting necessary reconstructed events for physics analysis.**

Event Metadata System

The Event Metadata System should have high performance, since it should be suitable to write such a big amount of metadata for all official reconstructed events saved to ROOT files and especially be concurrently accessed by a large number of jobs controlled by a Workload Management System, which are distributed on computing resources.

The number of events in the experiments will be sufficiently increased up to billions of events expanding its total capacity to terabytes. Moreover, the time of retrieving events for desired analysis shall be within acceptable limits.

Event Metadata System prototypes

Choosing a database management system (DBMS) for the Event Catalogue implementation, which ensures efficient processing of billions of records, is one of the key points of the design.

Comparative tests were carried out between various SQL and NoSQL DBMS :

- **PostgreSQL**
- **Apache Hadoop HBase / Phoenix**
- **Apache Cassandra**

Hardware Configurations

Hardware Configuration 1

XenServer hypervisor on a Dell PowerEdge FX430 server

- **2 x Intel Xeon E5-2680;**
- **RAM: 256 GB (240 GB is available) DDR4 2133 MHz;**
- **2 x Intel SSD SC1BG400G4R;**

  (only one SSD disk with the size of 400 GB is used for storing the databases, the second one stores only system information)
- **10 Gb/s Ethernet;**
- **software:**
  - **Scientific Linux 7.9,**
  - **PostgreSQL 12.5,**
  - **HBase 2.2.3, Hadoop 3.2.1.**

Hardware Configurations

Hardware Configuration 2

modern user desktop machine with no virtualization

- **Intel Core i9-10900F 2.80GHz (20 CPU cores);**
- **RAM: 64 GB;**
- **1TB NVMe SSD;**

- **software:**
  - **CentOS Linux 8.2,**
  - **PostgreSQL 12.5,**
  - **Apache Cassandra 3.11.8.**

Database structure

The databases have been filled with the same randomly generated data in the following ranges:

- **SOFTWARE_ID (INT, 2 BYTE) = 0…3.**
- **PERIOD_NUMBER (INT, 4 BYTE) = 1…7.**
- **RUN_NUMBER (INT, 4 BYTE) = 1…1000.**
- **EVENT_NUMBER (INT, 4 BYTE) = 0…200000.**
- **DETECTOR_HIT (INT, 4 BYTE) = 0…MAX_INT.**
- **PRIMARY_VERTEX (BOOLEAN, 1 BYTE) = FALSE…TRUE.**
- **IF PRIMARY_VERTEX IS FALSE:**
  - **PRIMARY_TRACKS (INT, 4 BYTE) = 0, ALL_TRACKS (INT, 4 BYTE) = 0…20;**

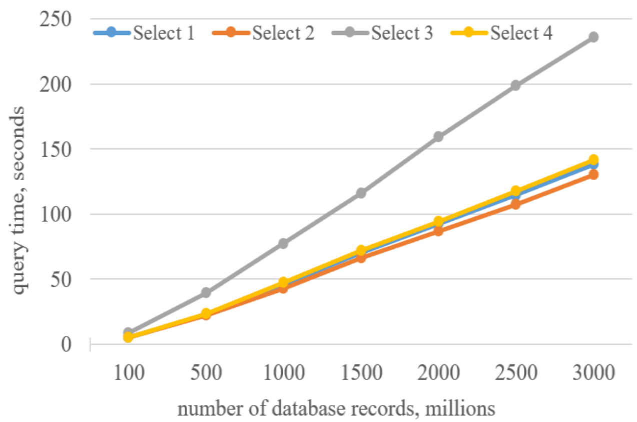
  IF PRIMARY_VERTEX IS TRUE:
  - **PRIMARY_TRACKS (INT, 4 BYTE) = 1…30, ALL_TRACKS (INT, 4 BYTE) = PRIMARY_TRACKS…300;**
- **POSITIVE_TRACKS (INT, 4 BYTE) = 0…PRIMARY_TRACKS.**
- **PARTICLES (INT, 4 BYTE) = 0…MAX_INT.**

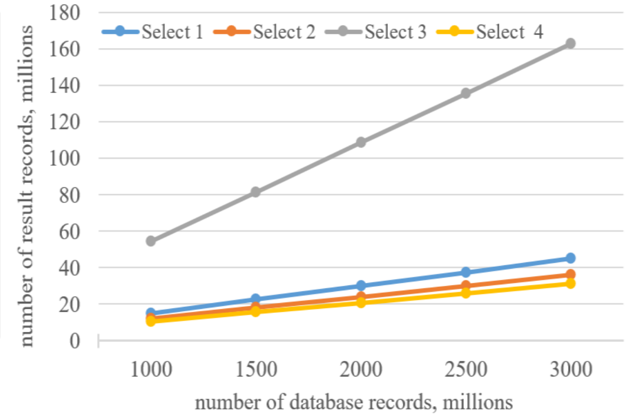
PostgreSQL

test queries:

- **SELECT 1: PERIOD_NUMBER=6 AND SOFTWARE_ID=0 AND PRIMARY_TRACKS > 5;**
- **SELECT 2: PERIOD_NUMBER=6 AND SOFTWARE_ID=2 AND PRIMARY_VERTEX = FALSE AND DETECTOR_HIT[BITS:0-5];**
- **SELECT 3: PERIOD_NUMBER=5 AND RUN_NUMBER > 100 AND PRIMARY_VERTEX = TRUE AND PARTICLES[BITS:5-9] > 4;**
- **SELECT 4: PERIOD_NUMBER=4 AND SOFTWARE_ID=1 AND PRIMARY_TRACKS>6 AND ALL_TRACKS>40 AND PARTICLES[BITS:10-14].**

PostgreSQL & Hardware Configuration 1

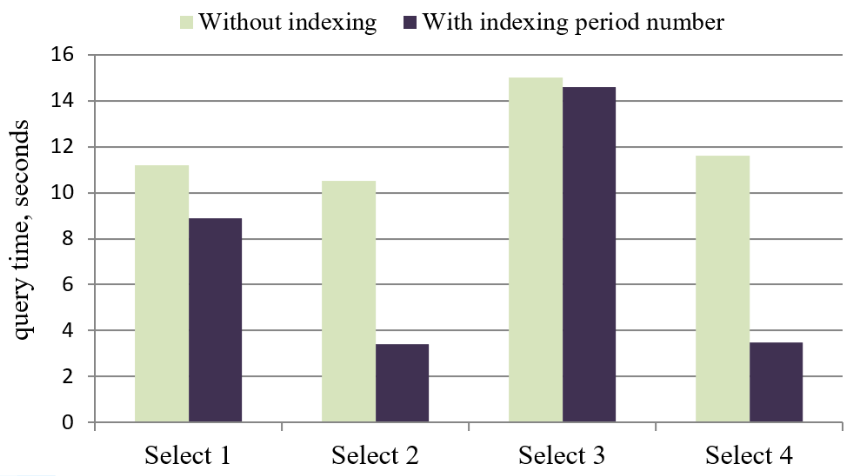




The left graph shows the dependence of the execution time of the test data requests on the number of event metadata records stored in the PostgreSQL database. It can be seen that the execution time of the queries is increased almost linearly. The execution time of Select 3 is much longer than the execution time of others.

The time difference can be explained by a sufficiently larger number of result records in Select 3, which is presented on the right graph.

PostgreSQL & Hardware Configuration 2



Linear increase, similar to the left graph in previous slide, of the query time for Select 1-4 has been obtained.

The query times for the same database size and records with Hardware Configuration 2 are lower.

Without custom indexing, PostgreSQL performs full table scans to select data based on filtering criteria. To check how indexing can improve the performance, a btree index over PERIOD_NUMBER attribute was created.

Hadoop HBase & Hardware Configuration 1

HBase runs on top of HDFS (Hadoop Distributed File System) and is designed to provide a fault tolerant mechanism of storing large collections of sparse data sets. HBase achieves high throughput and low latency by providing faster read/write access on huge datasets. Hence, it is the choice for applications that require fast and random access to large amounts of data.

In our tests we used *pseudo-distributed* mode configuration. Pseudo-distributed mode means that HBase runs completely on a single host, but each HBase net daemon runs as a separate process.

We also used Apache Phoenix. Phoenix is an open source SQL skin for HBase. When working with Apache Phoenix you use the standard JDBC APIs instead of the regular HBase client APIs to create tables, insert data, and query your HBase data.

Hadoop HBase & Hardware Configuration 1

Database configurations with HBase:

- **HBase C1.**

  One big table with all the event metadata, without Apache Phoenix.

- **HBase C2.**

  One big table with all the event metadata, with Apache Phoenix

- **HBase C3.**

  Event metadata are distributed among tables depending on a value of primary key fields (SOFTWARE_ID, PERIOD_NUMBER, RUN_NUMBER). The combination of the fields is unique for each table. Apache Phoenix is not used.

Hadoop HBase & Hardware Configuration 1

test queries:

- TEST 0: GET ALL RECORDS (SELECT COUNT(*));
- TEST 1: PERIOD_NUMBER = 6 AND SOFTWARE_ID = 0 AND PRIMARY_TRACKS > 5;
- TEST 2: PERIOD_NUMBER = 6 AND SOFTWARE_ID = 2 AND PRIMARY_VERTEX = FALSE;
- TEST 3: PERIOD_NUMBER = 4 AND SOFTWARE_ID = 1 AND PRIMARY_TRACKS > 6 AND ALL_TRACKS > 40;
- TEST 4: PERIOD_NUMBER = 5 AND RUN_NUMBER = 27 AND PRIMARY_VERTEX = TRUE;
- TEST 5: PERIOD_NUMBER = 5 AND RUN_NUMBER > 308 AND PRIMARY_VERTEX = TRUE.

Hadoop HBase & Hardware Configuration 1

	HBase C1 (without Phoenix) One global large table. 500 000 000 rows.	HBase C2 (with Phoenix) One global large table. 500 000 000 rows.	HBase C3 (without Phoenix) The collection of many tables, sorted by index. Cumulatively 500 000 000 rows.
Test 0	56 min	28 min	63 min
Test 1	29 min 55 sec	28 min 02 sec	5 min
Test 2	32 min 4 sec	28 min 47 sec	11 min
Test 3	30 min 20 sec	29 min 52 sec	8 min
Test 4	28 min	not supported by Apache Phoenix	1 min 52 sec
Test 5	29 min	not supported by Apache Phoenix	2 min 12 sec

Apache Cassandra & Hardware Configuration 2

Apache Cassandra is a distributed open-source NoSQL database management system. Cassandra stores data in tables that, for high performance, must be designed based on expected queries to the database.

In addition, primary keys must be specified for data organization in tables. There are two components of the Cassandra primary key: partition and clustering key (both could be single or compound).

The partition key determines the node on which a given data row will be stored.

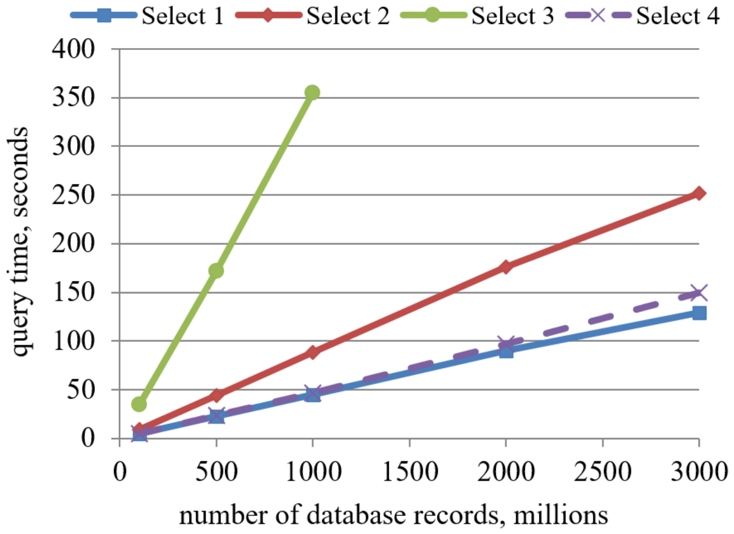
Apache Cassandra & Hardware Configuration 2

Most of the test requests (Select 1-4) contain specific values of PERIOD_NUMBER and SOFTWARE_ID, so these values are natural candidates for inclusion in the partition key.

Clustering keys can be chosen in different ways, but because Tests 1 and 4 use a PRIMARY_TRACKS comparison filter, PRIMARY_TRACKS was included as a first attribute of the clustering key.

As a result, generated test metadata were saved in a table with the following primary key: ((PERIOD_NUMBER, SOFTWARE_ID), PRIMARY_TRACKS, EVENT_NUMBER), where (PERIOD_NUMBER, SOFTWARE_ID) is a composite partition key, while PRIMARY_TRACKS and EVENT_NUMBER are attributes of the clustering key.

Apache Cassandra & Hardware Configuration 2



The selected primary key is well-suited for Select 1 and 4, which demonstrate fast processing times comparable with PostgreSQL times on the same hardware configuration.

Select 2 involves a full scan of event metadata in a given partition with specified (PERIOD_NUMBER, SOFTWARE_ID) key values, so it shows worse performance.

Select 3 requires a scan over multiple partitions, therefore its performance is expectedly quite low. It is important to emphasize that the slow performance of some of the queries is due to the specific primary key selection.

Conclusions

- **The performed tests of the database management systems for the implementation of the Event Catalogue have shown that the relational database built on the Postgres DBMS ensures an acceptable query execution time for selecting events on a modern hardware configuration and it is easy to configure;**

Conclusions

- The tested NoSQL database management systems are also able to demonstrate a good time of metadata retrieval. However, configuring the databases is much more difficult and a chosen structure might be suitable for some types of queries and shows poor performance for other queries for which it is not designed. Furthermore, for efficient work with NoSQL databases, it should be necessary to deploy and maintain a cluster of database servers.

Conclusions

- At the same time, further research on non-relational databases and their performance for implementation of the Event Database is planned to be continued on JINR distributed hardware resources.

Thanks for attention !