Introduction

The Trigger and Data Acquisition (TDAQ) system of the ATLAS experiment, the Large Hadron Collider (LHC) at CERN currently is composed of a large number of distributed hardware and software components (about 3000 machines and more than 25000 applications) which, in a coordinated manner, provide the data-taking functionality of the overall system.

During data taking runs, a huge flow of operational data is produced in order to constantly monitor the system and allow proper detection of anomalies or misbehaviors. The Persistent Back-End for the ATLAS Information System of TDAQ (P-BEAST) is a system based on a custom-built time-series database and it is used to archive and retrieve for applications any operational monitoring data. P-BEAST stores about 18 TB of highly compacted and compressed raw monitoring data per year acquired at 200 kHz average information update rate during ATLAS data taking periods.

How (strategy)

Alternatives

P-BEAST can store integers, floats, strings and arrays and structures of the above. It also has support for schema evolution/modifications.

InfluxDB:
- Time-series database
- Support for storing integers, floats, strings and no control over signedness and word size
- Schema-less
- High write speed (reportedly)

ClickHouse:
- Columnar database
- Support for storing integers, floats, strings and control over signedness and word size
- High write speed (reportedly)

Tested data models

InfluxDB #1
- Single table (measurement in InfluxDB terminology)
- Timestamp and a tag (InfluxDB indexed column) containing the object name make up the primary key
- InfluxDB: Arrays are stored using one column (field in InfluxDB terminology) for each element of the array
- For versioning purposes, the data types are stored as part of the column name for all tested data models

InfluxDB #2
- Multiple tables (measurements in InfluxDB terminology)
- Timestamp is the primary key in each table
- Each table contains data from a single object
- The object name is stored as part of the table name

ClickHouse #1
- Single table
- Columns containing the timestamp and the object name make up the primary key
- The object name column is used to create partitions (ClickHouse concept)

ClickHouse #2
- Multiple tables
- Column containing the timestamp is the primary key in each table
- Each table contains data from a single object
- The object name is stored as part of the table name

How (implementation)

Software

• The tested data models have each been implemented as a separate test that was run on 6 types of ATLAS operational monitoring data (see below);
• All the tests have been implemented in the Go Programming Language;
• InfluxDB is itself written in Go, so it comes with a Go client library from the developer;
• ClickHouse has multiple third-party Go libraries.

Hardware

• All the tests have been run on a dual-CPU machine with:
  • 2 Intel Xeon ES-2630 v2 @ 2.60GHz CPUs (each with 6 cores and Hyperthreading, for a total of 24 threads) and
  • 32 GB of RAM and
  • An 18 TB RAID0 array using hard disk drives

What

<table>
<thead>
<tr>
<th>Data Points / Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
</tr>
<tr>
<td>10000</td>
</tr>
<tr>
<td>100000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test (Data Type)</th>
<th>Single Integer</th>
<th>Single Float</th>
<th>Array of 12 Floats</th>
<th>Array of 3.5K Floats</th>
<th>Single String</th>
<th>Large Single String (~5.5KB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion Rates</td>
<td>InfluxDB #1</td>
<td>InfluxDB #2</td>
<td>ClickHouse #1</td>
<td>ClickHouse #2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Status and Outlook

• In the best case scenario, ClickHouse’s insert rate is faster than that of InfluxDB (all tests except the one for Single Integer indicate this);
• What happened with the Single integer test? The number of objects for that attribute was very large (over 14k) and ClickHouse’s partitions feature and feature for each partition make the read speed decreases significantly. It seems that the insert speed is negatively impacted as well;
• Single table approach almost always provide a better result;
• Future work: What happens when it comes to read speed? Or storage performance? How about those server startup times issues that were noticed? We will investigate and get back to you!

References