Evaluating InfluxDB and ClickHouse database technologies for improvements of the ATLAS operational monitoring data archiving

Matei-Eugen Vasile[1], **Giuseppe Avolio**[2] **and Igor Soloviev**[3]

[1]Institutul Național de Cercetare-Dezvoltare pentru Fizică și Inginerie Nucleară Horia Hulubei (IFIN-HH), Măgurele, România
[2]European Laboratory for Particle Physics, CERN, Geneva 23, CH-1211, Switzerland
[3]University of California, Irvine, CA 92697-4575, US

E-mail: matei.vasile@cern.ch

Abstract. The Trigger and Data Acquisition system of the ATLAS experiment at the Large Hadron Collider at CERN is composed of a large number of distributed hardware and software components which provide the data-taking functionality of the overall system. During data-taking, huge amounts of operational data are created in order to constantly monitor the system. The Persistent Back-End for the ATLAS Information System of TDAQ (P-BEAST) is a system based on a custom-built time-series database. It is used to archive and retrieve any operational monitoring data for the applications requesting it. P-BEAST stores about 18 TB of highly compacted and compressed raw monitoring data per year. Since P-BEAST's creation, several promising database technologies for fast access to time-series have become available. InfluxDB and ClickHouse were the most promising candidates for improving the performance and functionality of the current implementation of P-BEAST. This paper presents a short description of main features of both technologies and a description of the tests ran on both database systems. Then, the results of the performance testing performed using a subset of archived ATLAS operational monitoring data are presented and compared.

1. Introduction

The Trigger and Data Acquisition (TDAQ) system of the ATLAS [1] experiment at the Large Hadron Collider (LHC) [2] at CERN is currently composed of a large number of distributed hardware and software components. There are about 3000 machines and in the order of $\mathcal{O}(10^5)$ applications running at any given time, which, working in concert, provide the data-taking functionality of the detector 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 any potential anomalies or misbehaviors. The Persistent Back-End for the ATLAS Information System of TDAQ (P-BEAST) [3] is a system based on a custom-built time-series database. It is used to archive and retrieve for applications any requested 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.

2. Background

Since P-BEAST has been put into production, four years ago, several promising database technologies that feature fast access to time-series or column-oriented data become available. InfluxDB [4] and ClickHouse [5] have been the most promising candidates for use in improving the performance and functionality of the current implementation of P-BEAST.

A survey was done to identify if any suitable candidate technologies that could potentially be used in improving P-BEAST exist. Once viable candidates have been identified, a series of synthetic tests have been planned and ran on these database systems in order to try and leverage the best possible options for storage of the P-BEAST data using their respective data model capabilities.

These performance tests have been performed using a subset of archived ATLAS operational monitoring data.

3. Testing

3.1. Testing strategy

P-BEAST can store integers, floats, strings and arrays and structures of these data types. A preliminary survey was done amongst the existing time-series databases, columnar databases, key-value stores and anything else that could conceivably work as a fast backend for P-BEAST, were available on the market, and were, ideally, Open Source Software. Only two candidate database have been identified: InfluxDB (a time-series database) and ClickHouse (a columnar database).

One of the most important qualities in a candidate was raw write speed. A minimum write speed of 100k metrics/s was used as a point of reference. During our survey, we discovered an existing collaborative, community-effort, survey that has been keeping track of about 20 existing time-series database systems and their characteristics. This survey proved most useful because it gave us a starting point to easily identify which of the existing technologies met our minimum write speed reference criterion. Moreover, other qualitative factors, beyond raw write and read performance, were important in this preliminary research. The factors directly relevant to the write performance testing described here are concerned with the database systems' capability of mapping existing P-BEAST data on the data model supported by each database technology:

- support not just for numeric values but also for strings,
- support for arrays (or ability to simulate it) and
- support for structures (or ability to simulate it).

Using these data model requirements, we filtered the candidate pool we already had from the write speed survey and ended up with the final two candidates: InfluxDB and ClickHouse.

InfluxDB is a time-series database with support for storing integers, floats, strings. It is schemaless and has a high write speed according to the 100k metrics/s reference we have been using. InfluxDB organizes data using *measurements* (similar to the tables from classic DBMSes), *fields* & *tags* (similar to columns) and *points* (similar to rows).

The timestamp of a point is always the primary key. Tags are always indexed, fields are never indexed. The more tags are used, the lower the performance, but adding new fields has very little impact on performance. A collection of data that share a measurement and a tag set constitutes a series. The series cardinality is the number of unique measurements and tag set combinations in a database and is crucial for database's performance. Both tags and fields can be added in a measurement on the fly, making InfluxDB de facto schemaless.

ClickHouse is a columnar database with support for storing integers, floats, strings, arrays and control over signedness and word size. It also has a high write speed (same 100k metrics/s reference).

On the surface, at least regarding its query language, ClickHouse is much closer than InfluxDB to a classic relational DBMS. However, it lacks the schema flexibility of InfluxDB.

3.2. Tested Data Models

P-BEAST *objects* are stored using a *class.attribute* system. The smallest piece of data stored is a time series data point that represents a value of an *attribute*. The *attributes* whose values are stored belong to *objects*. The most important dimension is the number of *objects* that need to be stored. The *class* and *attribute* counts are less variable compared to *object* counts. If they grow, the database storage system can be easily scaled horizontally. If *object* counts increase, there is no way but for the database to handle the higher counts. Thus, the very important series cardinality will be linked to *object* count.

Drawing up the testing methodology started from the particularities of InfluxDB, because InfluxDB was the first database being tested. The focus of this first round of testing was write performance.

For versioning purposes, the data types are stored as part of the column name for all tested data models, both in InfluxDB and in ClickHouse. For example, if we have variable *x* of data type *integer* and at a version change the data type of *x* becomes *float*, we encode the data type using the field key/column name: before the version change we use field/column *x:int* and after the version change we use field/column *x:float*.

InfluxDB has no native support for arrays, so arrays are encoded using multiple fields and encoding the index of the array elements in the field keys. For example, if we have array *vector* of length *N*, we encode it using the fields named *vector[1]*, *vector[2]* ... *vector[n]*. Structures are encoded using multiple fields. For example, if we have structure *struct*, we encode it using the fields named *struct.element_a*, *struct.element_b* etc.

These rules can be combined to encode the types of the data stored into arrays (e.g. *vector[k]:int*, *array[n]:string* etc.) or structures (e.g. *struct.element:float* etc.). Furthermore, they can be combined even more to create structures of arrays, arrays of structures etc. (e.g. *vector[m].element_x:int*, *struct.array[y]:float* etc.);

InfluxDB #1 test: Single measurement - the timestamp and a tag containing the *object* name make up the primary key in each measurement (Figure 1). A P-BEAST *attribute* is stored using a single measurement, and the *object* tag is used for defining the P-BEAST *object* that data point belongs to.

InfluxDB #2 test: Multiple measurements - the timestamp is the primary key in each measurement. Each measurement contains data from a single *object*. The *object* name is stored as part of the measurement name (Figure 2).

ClickHouse has support for arrays. Because of it, in the ClickHouse tests arrays are stored in a single column. The rest of the data model is organized along the same lines as in the case of the InfluxDB tests. In order to maintain similarity with the InfluxDB setup, when using object name column are used, they are used to create partitions. However, the ClickHouse documentation recommends that *"you shouldn't make overly granular partitions (more than about a thousand partitions)"* [6].

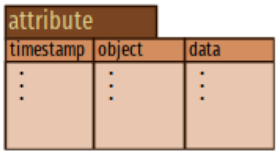

Figure 1. Single table setup

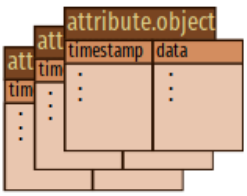

Figure 2. Multiple tables setup

ClickHouse #1 test: Single table - the columns containing the timestamp and the *object* name make up the primary key. The *object* name column is used to create partitions (Figure 1). A P-BEAST *attribute* is stored using a single table, and the *object* column is used for defining the P-BEAST *object* that data point belongs to.

ClickHouse #2 test: Multiple tables - the 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 (Figure 2).

3.3. Testing setup

3.3.1. Software The four tested data models have each been implemented as a separate test that was run on six types of ATLAS operational monitoring data. The monitoring data types have been selected in order to have as wide an array of data types as possible:

- single integer
- single float
- array of 12 floats
- array of approximately 3.5k floats
- single string
- large single string (approximately 5.5kB)

All the tests have been implemented in the Go Programming Language [7]. InfluxDB is itself written in Go, so it comes with a Go client library from the developer. The library is based on the built-in InfluxDB HTTP API [8]. ClickHouse has multiple third-party Go libraries. Some of them are based on the low level native ClickHouse TCP client-server protocol[9]. For performance reasons, we chose one based on the TCP protocol in order to avoid the extra overhead of the HTTP protocol. Moreover, the chosen library also has support for bulk writes [10].

All four tests have been developed to be as similar as possible, so that the comparison of the resuls would yield as clear conclusions as possible. They all follow the same pattern. All of them receive the *batch size* as an input parameter which will be used for the size of the bulk writes. Data is read into a prepared batch which is written to the database when the defined size or the end of the available data has been reached.

3.3.2. Hardware All the tests have been run on a dual-CPU machine with:

- 2 Intel Xeon E5-2630 v2 @ 2.60GHz CPUs (each with 6 cores and Hypethreading, for a total of 24 threads) and
- 32 GB of RAM and
- An 18 TB RAID0 array using hard disk drives

4. Results and conclusions

In most cases, the write performance of ClickHouse has been better than that of InfluxDB (Figure 3). The exception that is clearly visible in the case of the *single integer* data is caused by a combination of factors whose importance was not fully assessed in the planning phase of the testing. First, there is *series cardinality*. The InfluxDB documentation states that a single node, under high load conditions, should be able to handle over 250k field writes per second while having a series cardinality of over 1 million in the database [11]. When implementing the ClickHouse tests, the idea was to use ClickHouse's partitions feature in order to have a similar functionality to the InfluxDB series. However, as mentioned earlier, the ClickHouse documentation warns about not having a large

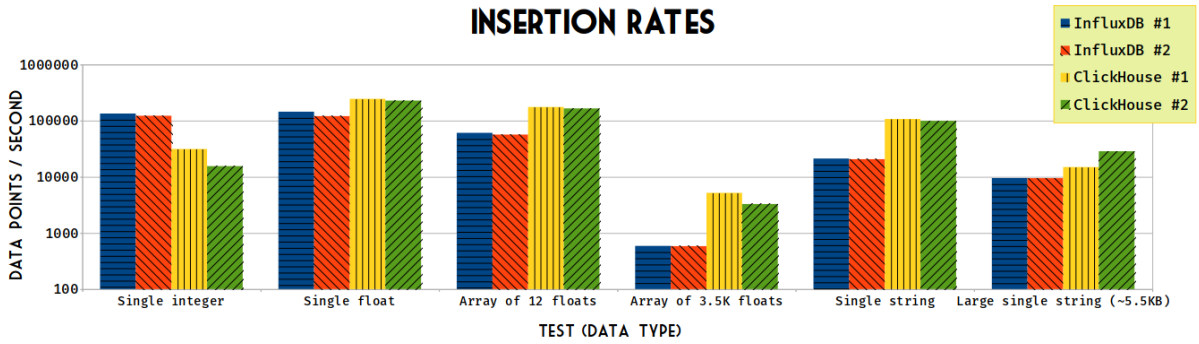

Figure 3. Results

numbers of partitions and definitely not going over 1k partitions, with the mention that SELECT queries would be negatively impacted. However, as it turns out, writes are also negatively impacted. The *single integer* data has a *series cardinality* of over 45k. As seen in the results, the ClickHouse write performance suffered noticeably because of it.

Also, the multiple table approach has almost always been slightly poorer than the single table approach. Taking into account that when using the multiple table approach there is a need to perform joins when retrieving results that use data stored in more than one table, at this point, it seems like that the single table approach is better.

4.1. Future work

Taking into account this first round of results, we will be designing a second round of testing that takes into account aspects what weren't taken into account in the first round. First, series cardinality and ClickHouse's partition count limitation will be taken into account. Second, only the single table approach will be tested, because all the results and analysis indicate that is the better approach. Furthermore, this second round of testing will be designed so as to be useful in drawing valid conclusions regarding storage performance as well.

Finally, after the second round of write performace testing, a series of test for assessing read performance will need to be developed.

References

[1] ATLAS Collaboration 2008 *JINST* **3** S08003
[2] Evans L and Bryant P 2008 *JINST* **3** S08001
[3] Avolio G, D'Ascanio M, Lehmann-Miotto G and Soloviev I (ATLAS Collaboration) 2017 A web-based solution to visualize operational monitoring data in the Trigger and Data Acquisition system of the ATLAS experiment at the LHC Tech. Rep. ATL-DAQ-PROC-2017-001. 3 CERN Geneva URL `https://cds.cern.ch/record/2242032`
[4] InfluxData InfluxDB URL `https://www.influxdata.com/time-series-platform/`
[5] Yandex ClickHouse URL `https://clickhouse.yandex/`
[6] Yandex ClickHouse Documentation: Custom Partitioning Key URL `https://clickhouse.yandex/docs/en/operations/table_engines/custom_partitioning_key/#custom-partitioning-key`
[7] Google The Go Programming Language URL `https://golang.org/`
[8] InfluxData InfluxDB HTTP API reference URL `https://docs.influxdata.com/influxdb/v1.7/tools/api/`
[9] Yandex Native Interface (TCP) URL `https://clickhouse.yandex/docs/en/interfaces/tcp/`
[10] Shvakov K GitHub - kshvakov/clickhouse: Golang driver for ClickHouse column-oriented database management system URL `https://github.com/kshvakov/clickhouse/`
[11] InfluxData General hardware guidelines for a single node URL `https://docs.influxdata.com/influxdb/v1.7/guides/hardware_sizing/#general-hardware-guidelines-for-a-single-node`