Big Building Data 2.0 - a Big Data Platform for Smart Buildings

Big Building Data 2.0 - a Big Data Platform for Smart Buildings

Lucy Linder[1], Frédéric Montet[1], Jean Hennebert[1], Jean-Philippe Bacher[2]

iCoSys Institute[1], *HEIA-FR*, Fribourg, Switzerland
ENERGY Institute[2], *HEIA-FR*, Fribourg, Switzerland

E-mail: lucy.derlin@gmail.com, frederic.montet@hefr.ch, jean.hennebert@hefr.ch, jean-philippe.bacher@hefr.ch

Abstract. The modern built environment is now connected. Multiple software and protocols are used in buildings of many kinds, thus creating a fascinating and heterogeneous environment. Within this context, applied research can be complicated and would benefit from a single data location across projects and users. The first version of BBData tried to solve this problem, BBData v2.0 is an update with a better-defined scope and a new codebase.

The solution has been open sourced and simplified with a full software rewrite. Its components are now state-of-the-art and proven to be stable in industrial settings. The achieved performances have been thoroughly tested. Together with its new architecture, BBData v2.0 now accommodates the needs of modern experiments; efficient for simple proof of concepts while keeping the possibility to scale up to city-level projects. This flexibility makes BBData a good candidate for research while being able to scale in production settings.

1. Introduction

In 2021, the built environment is connected. Smart homes are gathering data, but they are not the only ones. Key infrastructures, such as district heating networks, need to store and compute sensor data [1]. City pollution monitoring plans are creating constantly new data solutions to make sure that no citizen is left in a cloud of unbreathable air [2, 3]. The use cases are numerous and the technologies they use are heterogeneous.

For the single use case of smart buildings already many protocols are used. Still, there is no standard way to access data with a single tool for multiple buildings [4, 5].

In the context of applied research, an easily accessible data store has many benefits. For instance, it makes prototyping faster, data storage best practices are shared between researchers, multi-project data archives are created, novel research is fostered by interweaving multiple data sources and so on. In fine, the BBData endeavour is an attempt to bring such a solution with modern technologies to accelerate research with greater quality.

This paper is structured with a method section presenting the key findings from BBData in its first version and the improvements of the new version. Then, the result section analyses the software in its production environment.

2. Overview

2.1. BBData v1.0

The first version of BBData has been thoroughly explained in [6] and is summarized in the left part of figure 1. Sensors generate data (1), which are encoded into JSON records (2) that include a timestamp, a security token and the virtual object ID to which the sensor belongs. Those records are sent to BBData through a RESTful JavaEE application (3) running on a GlassFish server, the input API, which validates the token and stores the raw record into a a Kafka topic[1] (4) before returning a response. At the core of BBData run different Flink streaming applications, all connected through Kafka topics. A first one takes the raw record and "augments" it with metadata (unit, data type, ...) stored into a MySQL database (5). Another stores this augmented measure into a Cassandra NoSQL database (6). Optionally, other "processors" can read from Kafka and do side-processing, such as computing live aggregations. Users interact with BBData through another RESTful application, the output API (7), to manage the virtual objects, access the data and control the access rights in a fine-grained manner. The whole system runs on Hadoop [7].

2.2. Motivation behind a v2.0

The first version of BBData was driven by the idea of *Big Data*. This term has received many definitions, but all usually mention at least one of the following points [8]: the volume of data processed (Intel cites a median of 300 TB of data weekly), its complexity (unstructured data with complex transformations), and the technologies used (NoSQL, Hadoop, etc.).

Several years in production made it clear that the volume and complexity of the data at hand are not justifying some of our early choices. Parts of the pipeline thought to be critical could be simplified without a loss in throughput, while other parts could be improved by relying on more common technologies.

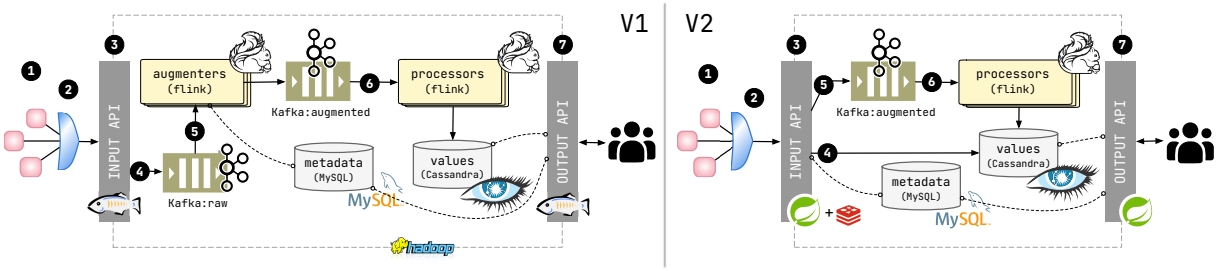

Figure 1. Overview of BBData v1.0 (left) and v2.0 (right)

3. Method

3.1. Main technology changes in v2.0

The second version of BBData keeps the same concepts with a simplified architecture and different technologies. An overview of the v2.0 architecture is shown on the right of figure 1.

First, the system is simplified by removing its dependency on Hadoop, which is difficult to maintain and requires a big infrastructure. Moreover, its main benefits could also be achieved through solid backup plans and good monitoring (see 4.2).

Second, the APIs are federated into one single executable jar, written in Spring Boot/Kotlin [9] and running on a light-weight Apache server built in the application itself. The notion of "profiles" is used heavily to let users control its runtime behavior, and optionally revert to the v1.0

[1] In this context, Kafka topics can be thought of as simple message queues.

Listing 1. Routes used to insert and retrieve data from an object.

```
POST /objects/values                          # Insert values

GET /objects/{objectId}/values                # Get value
    /objects/{objectId}/values/aggregated     # Get aggregated values
    /objects/{objectId}/values/latest         # Get latest values
```

model of having input and output APIs running on different machines. This switch simplifies the code, the maintainability, and benefits from the active SpringBoot community.

Third, the architecture is simplified by moving part of the processing to the input API itself. In v1.0, the input API returns a "200 OK" to the caller once the measure has been successfully submitted to the Kafka queue. However, this does not mean the data is safe or available at the other end. Moreover, if an error occurs in the streaming applications, there is no easy way to notify the caller, and the data may be lost forever. In v2.0, the raw measures are instead saved synchronously by the API (4). This adds some overhead, which is partially alleviated by the strategies described in section 3.2. Note that the input API still publishes augmented measures to Kafka (5), to support optional processing such as live aggregations.

3.2. Add a caching strategy and asynchronous updates

In v2.0, the input API now needs to pull metadata from the MySQL database and write to Cassandra. Some statistics must also be updated, as BBData keeps read/write counters for each virtual objects.

Cassandra and Kafka are extremely fast and made for this kind of use case. In comparison, MySQL is a "snail" and could become a bottleneck. At input time, MySQL must be queried first to validate a security token, then to retrieve the metadata associated with a virtual object (unit, type, etc). To alleviate part of this burden, BBData may use an in-memory Key/Value cache, either internal (HashMap), or external (Redis Cache). If enabled, the cached entries will consist of a key computed as `objectId:token`, which maps to the set of metadata for this object (or `null` if the token is invalid). As most of those data are immutable, the cached entries are only evicted in two cases: when a token is deleted, and when an object changes state (enabled/disabled). Note that in the latter case, the whole cache needs to be flushed, as only part of the cached key is known.

Updating statistics is also time-consuming. This is why by default, the input API updates the read/write counters asynchronously using a thread pool of configurable size and switches back to synchronous updates only if no more threads are available, thus ensuring no task is lost.

3.3. Routes and their documentation

The routes (URLs) from BBData have been reviewed to comply with the best practices of a common REST API. For instance, routes in Listing 1 allow users to insert and retrieve data with different variants. All routes are documented following the OpenAPI Specification [10] using in-code annotations. This ensures an up-to-date specification, which is used to generate an interactive documentation webpage and can be downloaded for use within software such as Postman [11].

Name	N	CPU	RAM	Disk	Services
cassandra	3	4	16	500	Cassandra nodes
web	2	16	16	50	API + Admin Webapp
db	1	16	32	256	MySQL + Redis
zookeeper	1	4	8	50	Zookeeper
zookeeper-kafka	2	4	8	50	Kafka + Zookeeper
flink	2	4	8	50	Flink worker // Flink master
monitoring	1	4	8	256	Grafana, SpringBoot admin, bbchecker
utils	1	4	8	32	NginX, rsyslog

Table 1. Overview of the virtual machines in the production cluster (Disk and RAM in GB).

4. Results

4.1. Minimal cluster

The second version of BBData can be deployed on a simple laptop. The API is a portable executable (`.jar`) and has hard dependencies only on MySQL and Cassandra, which can run on Docker containers[2]. Kafka and Flink are optional, and should only be used if side computations exist. The cache can be enabled without Redis, and in this case, a simple in-memory `Hashmap` will be used. This minimal setup is already quite resilient and should be able to support the equivalent of one to two smart buildings easily. As the need grows, the different components can be scaled horizontally.

4.2. Production cluster

BBData has been used in production for more than a year, under the supervision of the Smart Living Lab in Fribourg. The infrastructure is made of virtual machines provisioned, managed, and backed up through VMSphere. The base images are Ubuntu 18.04 LTS.

Table 1 shows the virtual machines composing the production cluster, and how services are distributed. The goal is to provide a strong infrastructure able to sustain the expected growth of the platform while limiting the hardware usage to avoid unnecessary costs. We wanted to ensure each machine could be restarted (or fail) without downtime. To achieve that, the API has been deployed twice on different machines, using Nginx as a load-balancer; Cassandra and Zookeeper run on three machines (quorum), and Kafka has two replicas.

To further monitor the live system, the tools Prometheus and Grafana have been set up, with custom Dashboards and alerting rules targeting Cassandra and the BBData APIs. Finally, a simple Python program triggered by Cron every 15 minutes interacts with the system (input, output), and sends a notification via instant messaging if an error occurs.

4.3. Stress Test

Stress Tests have run on the production cluster before going live. The first goal was to assess the impact of moving the virtual object statistics (read and write counters) from Cassandra to MySQL, the second to ensure the changes in v2.0 wouldn't impact the throughput and responsiveness of the system.

Stress tests were implemented using Apache JMeter. The *Test Plan* consists of three thread groups running in parallel, each testing a different capability: (1) a request to `/info` tests the API responsiveness and gives a baseline, (2) the submission of a new measure tests the input

[2] Dockerfiles with test data are available on GitHub at `https://github.com/big-building-data/bbdata-api`

	#threads	#requests		median (ms)	
		cass.	*sql*	*cass.*	*sql*
info	50	2,313	2,333	3	2
input	200	83,331	11,481	15	905
output	50	309,440	95,818	15	464
Total	-	395,084	109,632	15	468

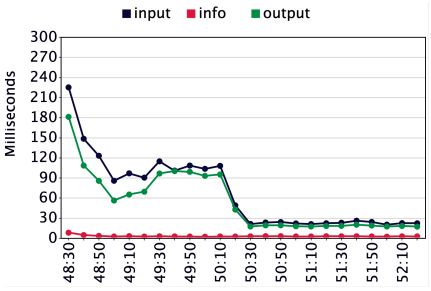

Table 2. On the left, median response time during 4 minutes stress tests on the production cluster using either Cassandra (*cass.*) or MySQL (*sql*) to store read/write counters. The threads column shows the number of concurrent threads doing requests. On the figure, evolution of those response times using Cassandra (*cass.*) over time.

API, (3) a request for the latest value tests the output API. Each thread in a group is configured to use a different object ID and waits 100ms between each request[3].

Results in Table 2 show that using MySQL for statistics slows down the system significantly: median response times are multiplied by 60 during submission, and 30 during query[4], which convinced us to stay with Cassandra. The graph on the right shows that the system needs some boot time before reaching stable performances (creating the cache, starting the threads, etc.). However, after 2 minutes, the response times stabilize around 15ms, a number confirmed by monitoring the live system.

4.4. Performances

Supposing sensors send an average of one measure per minute and given a response time of 15ms (see 4.3), the system is guaranteed to support 4K sensors. This baseline is improved by the fact that the API is not mono-thread, and that many types of sensors have a much higher average rate: weather sensors usually emit every 15min, presence sensors a few times per day, etc.

Moreover, most of the BBData components are scalable, either by design (Kafka, Cassandra, Flink) or through sidecar technologies (Nginx). The major bottleneck is MySQL, which may slow down significantly as the number of virtual objects and users increases. It is difficult to give a precise estimate of this limit, but we guess a change may be needed once the virtual objects reach the billion. The caching strategy described in 3.2 should help at least for the ingestion.

5. Discussion

Alternatives to BBData do exist. To cite a few: TimeScaleDB, InfluxDB or Amazon TimeStream. Nevertheless, the combination of time series storage with a permission system by project and user makes BBData ideal in a research context. For instance, it promotes the use of a single datastore across projects and institutes. Furthermore, its open source nature allows for in-house customization as well as de facto on-site data location; a usually sensitive topic for smart building data.

In terms of technologies used, the different updates allowed to transit towards a more attractive stack. All software components are state-of-the-art and have proven their reliability in industrial settings. The only potential drawback is MySQL for its lack of scalability, which could be fixed by the use of a distributed SQL database. Overall, with its simpler architecture,

[3] The JMeter stress plan is available on GitHub.
[4] Remember that counters are incremented asynchronously only if threads are available in the pool.

the system is now easier to maintain and can be tuned to the actual needs thanks to optional components.

To complete the software, however, ease of use without consultancy should be improved. Even if BBData itself proved its technical performances, its peripheral ecosystem is equally important and should be a priority. An interactive administration interface, a data visualisation platform, a great documentation as well as a single point of entry are all elements building a vibrant community, and are planned to be addressed in the near future.

6. Conclusion

BBData proposes an open source time series database together with a flexible permission system. The solution is a good fit for research purposes where data privacy and storage location is important. The new version is composed of state-of-the-art, stable technologies used in industrial settings. The simple architecture allows for great maintainability and is made to handle a wide range of use cases from a simple experiment to city level project with ease. The core of the software is stable and its reliability has been proven, although its peripheral ecosystem would need further development.

References

[1] Gao L, Cui X, Ni J, Lei W, Huang T, Bai C and Yang J 2017 *Energy Procedia* **142** 1829–1834 ISSN 1876-6102 proceedings of the 9th International Conference on Applied Energy URL https://www.sciencedirect.com/science/article/pii/S1876610217363270

[2] Kaivonen S and Ngai E C H 2020 *Digital Communications and Networks* **6** 23–30

[3] Shindler L 2021 *Environmental Technology* **42** 618–631 pMID: 31291821 (*Preprint* https://doi.org/10.1080/09593330.2019.1640290) URL https://doi.org/10.1080/09593330.2019.1640290

[4] Aguilar L, Peralta S and Mauricio D 2020 Technological architecture for IoT smart buildings *2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE)* pp 1–6

[5] Pritoni M, Weyandt C, Carter D and Elliott J 2021 Towards a scalable model for smart buildings Retrieved from https://escholarship.org/uc/item/5b7966hh

[6] Linder L, Vionnet D, Bacher J P and Hennebert J 2017 *Energy Procedia* **122** 589–594

[7] Shvachko K, Kuang H, Radia S and Chansler R 2010 The Hadoop distributed file system *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)* pp 1–10

[8] Ward J S and Barker A 2013 Undefined by data: A survey of big data definitions (*Preprint* 1309.5821)

[9] Spring Boot makes it easy to create stand-alone, production-grade Spring based applications. https://spring.io/projects/spring-boot access: 2021-04-27

[10] OpenAPI specification https://swagger.io/specification/ access: 2021-04-27

[11] Postman - the collaboration platform for API development https://www.postman.com/ access: 2021-04-27