Designing and implementing a Big Data benchmark in a financial context: application to a cash management use case

Lilia Sfaxi¹ · Mohamed Mehdi Ben Aissa²

Received: 3 August 2020 / Accepted: 26 February 2021 / Published online: 29 April 2021

© The Author(s), under exclusive licence to Springer-Verlag GmbH Austria, part of Springer Nature 2021

Abstract

This paper details the steps followed to benchmark a cash management platform of an investment bank using a generic benchmarking solution called BABEL. We highlight the modular design of BABEL, and present an evaluation methodology and best practices for its application on real world systems. The performance results collected with BABEL for the cash management use case enables to define the right tradeoffs in terms of consistency and availability, in a way that respects the service level agreements defined by the clients. On the other hand, we show that the overhead caused by BABEL's integration with the platform at runtime is very negligible.

Keywords Benchmark · Big Data architectures · Performance analysis · Financial systems · Large scale systems · Streaming platforms

Mathematics Subject Classification 68U35 · 68M14 · 68M20

1 Introduction

In the financial domain, information represents the main source of competitiveness and income for businesses [1]. Having the most reliable information as soon as it's available means you are ahead of your competitors. Also, reusing data from several sources in a unified storage, homogenizing it and performing parallel computations on massive amounts of historical data is a huge advantage for financial companies. Indeed, modern financial companies are using data characterized mainly by the 3Vs

✉ Lilia Sfaxi
lilia.sfaxi@insat.u-carthage.tn
Mohamed Mehdi Ben Aissa
mehdi_benissa@yahoo.fr

¹ INSAT, University of Carthage, Tunis, Tunisia

² Tunisia Polytechnic School, University of Carthage, Tunis, Tunisia

(Volume, Velocity and Variety) [1], but also by the fourth V, *Veracity* [2], which reflects the impact of noise and biases on the quality of the data. This property is all the more important for financial data, especially in a benchmarking context, as the generated data must respect the characteristics of the real data.

A study conducted on the US market [3] shows that the number of financial transactions noted a 50 times increase in only a decade. Even before the outbreak of the COVID-19 pandemic, and due to the noticeable increase of mobile payments and e-commerce, the size of the payments market was predicted to reach 2 Trillion US\$ by the end of 2025 [4]. But volume is not the only issue for financial data. The need for a low latency when processing financial queries is at its peak, as a constantly increasing amount of trades is executed online [5]. In fact, nowadays, the financial market can experience up to 500 quote changes and about 150 trades in 1 ms [1]. Aside from that, financial data is composed of structured data, mostly in the form of time series, such as those provided from global stock exchanges, and unstructured data, gathered from social media, web articles and videos, that are becoming valuable sources in modern trading strategies [3].

The characteristics cited above justify the need for Big Data tools, methods and best practices, in order to extract meaningful values from raw data in a timely fashion. However, and despite the awareness of the decision-makers about the importance of this technological shift, it is still a struggle for most of them [6]. Indeed, financial companies have resorted for a long time to high performance computers to deal with intensive processing [5]. But the constant increase of data volume generated the need to scale out the system, and thus use new storage and processing paradigms. This shift comes with a whole set of challenges, and calls for the use of Big Data analytics as well as transactions' management.

Big Data platforms are beneficial for the financial market in many aspects [5]. They can help organizations add a layer of intelligence to their trades, by exploiting all available data in a reasonable time. In addition, they make it easy to avoid indistinguishable risks, give more accurate customer profiling and detect fraud in real-time.

Enforcing a robust data governance to safely use the data while maintaining the usability is a key success factor for these types of applications. On the other hand, integrating accurate structured data with high-volume social networks insights can reveal hidden market opportunities, contrary to more classical means [1].

With the increasing number of Big Data solutions developed in the financial community, the need for their evaluation and measurement arises [5]. The evaluation of Big Data systems can be done both at development time, in order to choose the right architecture to apply, and at deployment time, to monitor the data flow once the application is up and running. It is even more important and critical in the financial context. In fact, due to the requirements of the CAP theorem [7], that states that consistency, availability and partition tolerance cannot be applied simultaneously, developers and architects are faced with the difficult task of finding a reasonable tradeoff between the performance of the system and the correctness of the data. The 4V properties of Big Data encourage in general to favour availability over complete consistency. But in financial applications, and especially in the cash management context, this can be problematic, as consistency is a crucial requirement. This justifies the need for an evaluation system that helps to extract valuable metrics for an appropriate decision-making.

We opt for benchmarking as an evaluation method. A benchmarking platform for a financial application needs to be precise, scalable, adaptable, with a very low overhead and minimal costs.

Even though the literature is very rich in terms of Big Data benchmarking systems, very few works respect the features cited above. Many solutions [8–14] target specific application types, while others [15–24] are designed for certain technology stacks. As for the specific case of financial systems, very few papers offer a benchmarking platform [3,25], and even when they do, they lack the needed genericity and ease of integration.

In this work, we show how we used our generic benchmarking platform called BABEL to evaluate and validate a Big Data platform for the Cash Management of an European Corporate and Investment Bank.

Research methodology In order to present our contribution, we follow the Design Science Research Methodology (DSRM) applied to Information Systems research [26]. This methodology presents the recommended steps to follow in a research work according to the used approach adopted by the researchers. In this paper, the problem statement is formulated based on an observed use case problem found in a real project. We thus use the *Client/Context Initiated Solution*. Our process is structured as follows:

- STEP 1- **Problem identification:** The cash management use case presentation and identified issues.
- STEP 2- **Objectives of the solution:** What are the characteristics needed in a benchmarking solution for financial systems?
- STEP 3- **Design and development:** What is the architecture and implementation choices of the benchmarking solution?
- STEP 4- **Demonstration:** How was the benchmarking solution applied to the cash management use case?
- STEP 5- **Evaluation:** What are the obtained results of the evaluation?
- STEP 6- **Communication:** How do we plan on marketing the solution?

2 STEP1—problem identification: the cash management application

2.1 Presentation of the cash management use case

For several years, banks have been working on cash management international solutions servicing both international trade and transaction banking product lines [27]. In this context, the proposed cash management platform focuses on the transformation of the operational model of cash management using Big Data technologies.

This platform enables decision makers to form an opinion in a timely fashion, while taking into consideration the financial context where the competition is increasingly harsh. In addition, it makes real time transactions' management possible thanks to transactional modules supporting high data volume with low latency. It also aims to modernize the decision-making with real-time and streaming analytics for fraud detection and KYC (Know You Customer) modules.

Fig. 1 Cash management decisional platform architecture

Respecting the Service Level Agreements (SLAs) requires real-time validations of B2B important transactions. In parallel, KYC reports are generated on-demand, based on high data volume.

2.2 Architecture and implementation

The cash management platform, presented in Fig. 1, is divided in two sub-platforms. The first one is fully operational and aims to manage the different B2B transactions with an event processing approach. The second one is decisional with two different layers:

1. Fraud detection module: a stream layer that helps to validate the different transactions managed by the operational platform.
2. KYC (Know Your Customer) module: stores and analyzes data collected in real-time from internal and external sources, using change data capture technologies.

Different types of data sources are identified: the operational and transactional internal systems, social media and web crawling services that provide important information about different customers, internal accounting services, bank referential, internal CRMs, and various logs and metrics.

A big part of the legacy system is implemented using MQ-Series, representing operational transactions deployed all over the world. The jobs are done in real-time thanks to Apache Storm, an event processing technology that helped to integrate real time transactions into the decisional platform. Thanks to these jobs, transactions can be consumed immediately by the fraud detection modules. Other in-house collection jobs were implemented with Apache Spark and Apache Storm in order to integrate the rest of the datasets from both external and internal data sources.

Apache Kafka is used as a centralized message bus for event collection. It is also the point of contact between the different modules and sub-platforms. Once the data is collected into Kafka, Spark batch and streaming APIs are used for processing.

In the fraud detection module, Apache HBase is used to persist different states of the Spark Streaming jobs and referential with a high consistency level. This module, implemented with Spark Streaming, checks the conformity of the B2B transactions by verifying if one of the actors is blacklisted or has any fraud proof from social media and official institutions. It is designed to support real-time interactions with the operational platform with a high scalability level in order to respect the real time aspect of the transactional platform.

In the second module, raw data is stored into HDFS, then consumed by Spark batch jobs implementing prediction algorithms, OLAP queries and customer behavior analytics.

Once the data is collected into the data lake, KYC analytic jobs consume it. In this module, data is prepared for OLAP queries via Spark then presented using data visualization tools like Tableau or QlikView. Machine learning algorithms are implemented for a better customer behavior understanding, by combining data from different datasets.

2.3 Problem statement

The problem when we have to deal with complex Big Data architectures is not only their implementation but also their evaluation, especially for Big Data platforms including several layers and multiple technologies.

Designing and implementing such platforms requires performance and service level agreements validation as a high priority. In this context, the cash management platform is quite complicated to evaluate because of the lack of a generic benchmarking tool that helps to generate test data with adapted workloads and an end-to-end evaluation of its multi-layered architecture. This ascertainment triggered the need to implement an end-to-end solution that is at the same time generic and that easily integrates with complex systems.

3 STEP 2—objectives of the solution

Several works focused on the criteria that must be found in a benchmarking system to be considered successful. Han and al. [28] cited four necessary non-functional requirements for benchmarkability: *usability*, *fair measurement*, *measurability* and *extensibility*. On the other hand, Wang and al. [29] defined six requirements for Big Data benchmarking systems: (1) their ability to measure and compare Big Data systems and architectures, (2) their support of data with the 4V dimensions: Volume, Variety, Velocity and Veracity, (3) the diversity and representativeness of their workloads, (4) their coverage of representative software stacks, (5) their support of state-of-the-art techniques and the inclusion of emerging systems in different domains, and (6) their usability, a redundant requirement, that states that the solution should be easy to use, deploy and configure by an average user.

In another publication [30], the authors insisted that workloads must be *relevant*, which means that they have to cover the typical behaviors of the evaluated system,

portable, able to execute on different software systems and architectures, and *scalable*, adaptable to systems of different loads.

In order to execute a performance evaluation of the cash management system, we need to use a benchmarking solution that:

- Provides a set of custom and standard end-to-end performance metrics such as throughput, latency, power efficiency, and client CPU/memory consumption, under custom test cases.
- Runs in parallel with the system, without interfering with its behavior, and with negligible overhead.
- Adapts to any number of layers in the Big Data application, as well as to any type of data and workloads.
- Is able to benchmark streaming and batch processing systems, as well as operational and decisional platforms.
- Evaluates the behavior of the whole system, by measuring application- and system-level metrics.
- Offers a real-time reporting layer, that illustrates the system's metrics on the fly.
- Is configurable, and offers a scalable collection, processing engines and storage, in order to enable the continuous archiving of metrics for a better analysis.
- Is flexible: new layers, metrics, and processing jobs can be added during the different integration phases and at run-time.

We proceed in the next section with a state-of-the-art study, in order to see if existing solutions respect all of these constraints.

4 Literature review

When it comes to defining Big Data solutions for the financial discipline, and despite the obvious need, companies are still at a very early stage [31]. Most of the efforts are targeting customer outcomes [32], while neglecting other aspects that can get the best of the collected data, thanks to advanced analytics, natural language processing and video analysis [31]. Indeed, several possible applications can be identified, such as retail banking, credit scoring, algorithmic trading, risk management and regulatory compliance [1].

4.1 Big Data and benchmarking solutions in the financial context

Several works tried to harness the power of data in financial contexts. Some of them tried to get the best out of their textual data [33,34], but also from other sources such as social media or financial reports. The works covered several domains, such as fraud detection [35], stock price movements prediction [36,37] or the going-concern prediction [38].

These works show an interesting aspect of value extraction from available data. However, they do not use Big Data systems and platforms for this purpose, which represents our main concern. On the other hand, several works use Big Data solutions in financial projects. Zhong et al. [39] propose a Big Data approach to compute Volume-

synchronized Probability of Informed Trading (VPIN). uCash [40] is an ATM cash management system that performs fast analytics to provide insights from huge historical data and streams. QuantCloud [3] uses a modular design approach that collects market data and performs complex processing to extract dependencies. Stockinger et al. [25] present ACTUS, a Spark-based solution performing large-scale risk analysis. Li et al. [41] designed an information management system for regulating the use of personal informations of financial consumers, with a strong focus on security.

Some of these Big Data solutions add a benchmarking layer to evaluate their performance. Zhang et al. [3] developed a benchmark to analyze the performance of QuantCloud, and Stockinger et al. [25] compared diverse implementations of the ACTUS framework.

These benchmarking solutions target specifically the developed application. Very few platforms, such as BigDataBench [29], propose a generic solution for data generation and performance evaluation that can apply to any financial application. But this solution, even if it is technology agnostic, does not propose an end-to-end performance monitoring and does not support the evaluation of multi-layered applications. Its main advantage is to propose a very varied set of workloads and a restricted integration with a set of specific technologies rather than a full and end-to-end architecture benchmarking and performance monitoring.

With the lack of benchmarking solutions that apply to the specific use case of financial applications, we looked for more general tools to be adapted to our project.

4.2 General benchmarking solutions

Big Data Benchmarking is not a new issue, and many solutions to collect performance information from Big Data systems are proposed in the literature [2,28,30,42,43]. They are mainly categorized as *technology-specific* and *application-specific*.

Technology-specific solutions A large number of these tools target the Hadoop environment and ecosystem. The ALOJA benchmarking platform [8], for instance, is used to benchmark the Hadoop environment when varying the system's architecture. MRBench [9], HiBench [10] and MRBS [11] are benchmarking suites destined to test MapReduce workloads while TPCx-HS [12] defines a standard produced by the Transaction Processing Performance Council (TPC) for benchmarking Hadoop runtime, API and MapReduce layers. PigMix [13] tracks the performance of the Pig query processor and SparkBench [14] targets all layers of the Spark framework.

Application-specific solutions This category groups technology-agnostic solutions, that specifically target certain types of applications. For example, BigBench [15], an application-level Big Data analytics benchmarking suite, captures operations via SQL queries and data-mining operations to be analysed via Batch-oriented processing. XBench [16] is a benchmark that targets XML databases. Chronos [17] is a framework for generating and simulating streaming data. StreamBench [18], proposes a middleware between the data generator and the data processors. Several works [19–22] focus on graph processing and storage, while others [23,24,44] specifically tackle Big Data solutions in the cloud.

Fig. 2 Architecture of BABEL

Discussion All the benchmarking systems cited above fail to offer a generic design that can be applied to any Big Data system and architecture, and more specifically in the financial context. On the other hand, most of these benchmarks evaluate individual system components (either hardware components like the CPU or networking, or software components like file systems), specific systems or operations [30] and must be combined or modified to be applied to Big Data applications with complex architectures. To the best of our knowledge, very few existing solutions are architecture-level, such as the one presented by Persico et al. [45], which compares the performances of Lambda and Kappa architectures for Online Social Networks. Even though this work focuses on the impact of the used architecture on the performance of the system, it does not provide a general design for benchmarking Big Data architectures, that is dynamic, agile and adaptable to any architecture with any number of layers.

5 STEP 3.1—design of BABEL

BABEL,¹ the **Big dAta BEnchmarking pLatform**, is a generic, scalable, distributed and end-to-end benchmarking platform for Big Data architectures. It defines two main layers as shown in Fig. 2: the *Benchmark Core Layer*, destined for the orchestration of the evaluation process, and the *Benchmark Integration Layer*, in charge of the communication with the system under test.

5.1 Benchmark integration layer

This layer's goal is to inject generated data to the system under test and to collect the performance metrics from its endpoints.

¹ The name is inspired from the myth of the Tower of BABEL, used to justify the origin of the presence of multiple spoken languages.

System under test (SUT) The SUT is the physical implementation of the architecture that we want to benchmark, which is the cash management application in our case. It can include an unlimited number of layers and technologies with different levels of dependency, distribution and service agreements. BABEL is compatible with different types of workloads: transactional, decisional or mixed, easily customizable by the tester. BABEL can also support other advanced and complex multi-layered architectures, such as Business Intelligence, Complex Event Processing, Data Science, or Internet of Things systems. In addition to that, it ensures a full support of cloud native applications and platforms.

Data generator According to Han et al. [28], using real data in benchmarking is not always recommended, from a security point of view. On the other hand, it can be really hard to replay these data in the right chronological order with a realistic frequency. This is why, most benchmarks [9,18,19,44] use data generators for performance monitoring, which guarantees the respect of Big Data constraints: high data volume, velocity and variety.

The data generator is in charge of generating test data for the benchmark. The tester can manage the generated data flow by defining several configuration parameters for:

- **Generated data:** number of fields, field length, etc.
- **Producers:** parallelism level, number of threads per producer, etc.
- **SUT:** connection parameters, hosts, ports, etc.

The data generation module is an extension of the Yahoo! Cloud Serving Benchmark (YCSB) [44]. It includes several Big Data connectors and gives a very high flexibility level, while integrating real use cases and a functional data generation.

As presented in Fig. 2, the data generation logic is managed by the orchestration layer and executed by the producers.

Benchmark producers and consumers The benchmark is connected to the SUT thanks to a set of scalable and distributed Producers and Consumers. The producers are agents that help disseminate data streams to the entry points of the system. The consumers are in charge of collecting the metrics at the output of each one of the SUT's layers.

The producer and consumer's API can be enriched in order to respect the target business and technical logic. The API is generic and can be configurable to have a specific Producer's or Consumer's personality.

Consumers have also the same flexibility level and can be deployed using multiple approaches:

- **Agent mode:** the default mode. Just like the producers, consumers are deployed via agent processes that listen to specific layers and send metrics to the benchmark store.
- **Embedded mode:** in this mode, the consumer is deployed in the SUT layer via the API or using other techniques, such as the Aspect-Oriented Programming (AOP).
- **Plugin mode:** this mode is used when existing metrics are available in the SUT by default and can be integrated in the benchmark store via specific plugins.

All these proposed modes have very little impact on the SUT performance, as all operations are asynchronous and based on an event approach, with a unique and centralized network time protocol. This is demonstrated in the Sect. 8.2.1: benchmark overhead and processing time.

5.2 Benchmark core layer

The benchmark core layer includes the main components of the benchmark, in charge of managing the life-cycle and configuration of the integration layer. This layer doesn't depend on the SUT and target workloads, as all its components are generic and extensible.

Benchmark orchestrator This is the bandmaster of the system, in charge of triggering all the components with the adequate configuration. The orchestrator is in charge of the data distribution between the producers and their consistency with the metrics sent by the consumers.

Benchmark store This component is a scalable storage system, in charge of storing all the events and metrics collected by the consumers.

Benchmark validator This component consumes the events from the store and transforms them into significant metrics and KPIs in order to prepare the next phase and simplify the reporting process.

Benchmark reporter The reporter generates convivial graphs and reports for the tester, to help the decision-making process and enable a user-friendly display of the performance metrics.

5.3 BABEL internal execution scenario

The internal execution scenario of BABEL is illustrated in Fig. 3. The main component and the entry point for the benchmark is the *orchestrator*, which has the responsibility to orchestrate all the other framework's components and to initialize the workflow with the appropriate configuration parameters (step 0). It first triggers the integration layer by sending the parameters to the *Data generator* (step 1). This component has the responsibility to load the configuration from the orchestrator and to delegate to each producer a test dataset based on time series as metadata for metrics traceability. Producers will then inject data to the SUT (step 2). In parallel and for each layer of the system under test, the consumers are managed by the benchmark orchestrator to consume results from the appropriate layer.

All integration components send asynchronous events (step 3 and 4) to the distributed *Benchmark store*. The *Benchmark validator* pulls the new stored events (step 5), and performs a real-time validation of the performance metrics (throughput, latency ...) (step 6), to be able to send the results to the *Benchmark reporter* (step 7). The latter will start by indexing all the KPIs and metrics (step 8), to finally generate real-time and dynamic reports.

6 STEP 3.2—implementation of BABEL

6.1 Technical architecture

We used Big Data technologies (Fig. 4) for the implementation of BABEL in order to handle the complexity of the SUT, the volume of the generated data and the need of

Fig. 3 Components interactions and sequence diagram

real-time processing in the validation and reports generation. Indeed, the Benchmark Store has to be horizontally scalable and sustain a high throughput and a low latency. The Benchmark validator, on the other hand, needs to perform real-time validation of the event flow received from the store.

BABEL has a modular architecture and is composed of generic modules that can be adapted to different use cases independently of the SUTs, thanks to the use of interfaces and abstract classes.

You may find the open-source code of BABEL in the following Github repository: <https://github.com/mehdibn/BABEL>.

Fig. 5 Physical architecture of BABEL

6.2 Physical architecture

In Fig. 5, we detail the physical architecture of BABEL and its integration with the SUT. We show the physical separation between modules that need to be deployed on different nodes. Each node is represented with a rectangle in the figure. We recommend, in a production setting, to get the best out of the available resources by distributing and parallelizing the components as much as possible. However, some components can be deployed in the same node when performance or capacity segregation are not needed between them.

6.3 Data generation and supported workloads

Data generation BABEL supports customized data generation that testers can enrich easily in order to simulate their specific business logic.

The default benchmark generator can be used for performance testing when random data processing is supported by the SUT. Many parameters can be configured in the BABEL configuration file, such as the number of records to generate, the number of fields per record, the length of field, etc.

Supported workloads BABEL offers the following workloads:

- **Write only [DEFAULT]:** can be used in order to simulate ETL systems, transaction systems, data collection, etc.
- **Read only:** can be used for user profile caching, when they are constructed elsewhere.
- **Update heavy workload:** a mix of 50% reads and 50% writes, and can be used for example for session storage and recording recent actions.
- **Read-modify-write:** in this workload, the client will read a record, modify it, and write back the changes. It can be used for example in user databases, where the records are read and modified by the user, or to record his activity.
- **Customized workloads:** can be configured by defining the proportion of each operation's type. In case of very specific workloads, testers can extend the **Workload** abstract class.

Both batch and streaming workloads are supported and can be configured by adjusting the number of records to generate, as well as the generation rate, that can be unbounded in order to simulate streams.

6.4 Getting started with BABEL

In order to use this framework to benchmark a Big Data architecture, we recommend the following approach :

1. Install Ansible on a separate node dedicated to the benchmark core and deploy its *ssh key* on the SUT nodes.
2. Clone the BABEL project from github: <https://github.com/mehdibn/BABEL>.
3. Check if the default data generator, producers and consumers are enough for your needs, or develop your own adapters.
4. Create your inventory describing the SUT and BABEL nodes.
5. Specify the configuration and architecture parameters in the *group_vars* files by defining:
 - (a) Workload: parallelism level, number of fields and records, etc.
 - (b) Data generator, consumers and producers configuration
 - (c) SUT environment: servers addresses, number of layers, resources name like tables, topics, queues, etc.
 - (d) BABEL installation parameters
6. White-test the framework by running the various components without actually calling a real SUT, and simulating its behavior using dummy events. This will measure the cost of basic benchmark communications.
7. The reporting dashboards will be automatically generated based on the SUT parameters defined by the tester in the configuration files.
8. Deploy and start the different benchmark components by using the existing Ansible playbooks.
9. Real time automatically-generated reports and dashboards will be refreshed in order to show the pre-defined metrics.

7 STEP 4—demonstration: application of BABEL in the cash management use case

7.1 Evaluation methodology

In order to evaluate the performance of the system, you first need to answer the following question: do we need to perform a load test (with a predefined reasonable workload) or a stress test (with an unbounded workload)? BABEL is configurable at runtime to support both approaches, depending on the needs of the tester.

Once this objective is fixed, testers need to answer the second question: what type of performance measures and metrics do we need to collect? In this perspective, different test scenarios can be considered:

- For a *performance tuning* purpose, it is advisable to maintain the same system and application configurations and to change one SUT parameter at a time to test its impact, such as: consistency level, partitioning, availability, memory, etc.

Fig. 6 Integration of the cash management use case using the Kappa architecture

- For a *scalability or resilience testing* purpose, we can scale up and down the SUT by adding and removing nodes to its clusters.
- To test the *deployment environment*, we recommend maintaining the same architecture, while deploying it to various environments, that can be cloud-based or on premise.
- To test the *performance of the chosen technologies*, we can keep the same architecture while switching from one technology to another.
- To evaluate and compare *multi-layered architectures*, we can keep the same technology stack but change the way they are composed and how they communicate.

BABEL is developed in a way that makes the benchmarking environment almost oblivious to the changes in the SUT. In fact, you just need to adapt the necessary producers and consumers by extending the existing APIs.

7.2 Benchmarking the system

In this section, we benchmark the fraud detection module of the global cash management platform presented in Fig. 1. This module is implemented with the Kappa Architecture.

The Kappa architecture [46], as presented in Fig. 6, combines the speed and batch features into a real-time layer. In fact, this layer is not only used for real-time stream processing, but also to capture historical data, recent and old, such as the clients' transactions for validation. And since there is no batch processing layer, only streaming jobs need to be maintained, which can decrease the complexity of the system.

Apache Kafka is deployed for low latency data ingestion. Once the transactions are collected, the speed layer, via Apache Spark Streaming, consumes them and executes a validation process by interacting with the historical data store and serving layer both implemented with Apache HBase via different tables. The biggest challenge in this module is the consistency, by insuring the *exactly once* semantics that will avoid transactions loss and their duplication. This feature is essential in a cash management

context that requires high consistency and integrity levels, but without significantly impacting the platform's performance.

In this regard, one of the challenges of this system is to adjust the performance/consistency ratio to enable a high consistency level with a very acceptable performance.

We aim to validate two metrics. First, scalability tests are done in order to validate the number of workers that can support the target workload. Test results show that at least 8 workers with 512 GB of RAM, 64 cores and 10 SSD disks of 2 TB for each one are needed in order to respect the required response time defined in the service level agreement (SLA).

In a second phase, another set of tests is implemented to measure the performance impact when tuning the consistency level. We define different values for Kafka and HBase replication factor, the producers number of acknowledgments per write operations and Spark checkpoints frequency.

In this paper, we focus on the second set of tests to give an idea about how BABEL can help to adjust the consistency level by respecting the announced performance objectives.

Data generation is customized as described in the implementation section in order to support the transactions format with a streaming approach and a specific business logic. For this purpose, the **lastValue()** and **nextValue()** methods of the **Generator<V>** abstract class are overloaded. We use a write-only workload in this case.

For Kafka and HBase, JMX (Java Management Extensions) consumers are deployed via the agent mode with Jolokia and Metricbeats. On the other hand, Spark jobs integrate embedded consumers by extending the **GenericConsumer** class.

8 STEP 5—evaluation of BABEL

8.1 Industry requirements and goals achievement

BABEL is designed to fulfill a set of industry requirements for Big Data benchmarking systems, presented in the objectives section. We show here how did BABEL satisfy them.

- **Genericity, adaptability and end-to-end approach:** thanks to the adapter's logic, BABEL can be integrated to any technology just by extending the generic consumers' and producers' API. Indeed, BABEL is agnostic to the technology used in the SUT. The fact that the number of consumers is unlimited and can support multiple layers of the SUT gives BABEL the ability to integrate with any architecture independently of its complexity. BABEL proposes the same approach for data generation and supported workloads. It implements their APIs in order to customize the test scenarios for a better integration with the applications.
- **Dynamicity, distribution and scalability:** The benchmark integration components were developed in order to minimize the overhead of BABEL. From an architectural point of view, the scalability and distribution of BABEL, insured

thanks to the use of Big Data tools, helps to absorb very high workloads from data generation to the reporting layers.

- **Multi-paradigm:** Existing or customized workloads can be easily defined for a better business integration. Thanks to this approach, both operational and decisional platforms with batch or real time engines can be benchmarked with BABEL.
- **Real-time:** All BABEL components are real-time and use asynchronous operations in order to make the final graphs and dashboards dynamic for a better restitution and decision making.
- **Agility:** BABEL supports SUT updates by including new layers and workloads without any impact on the existing components. Its architecture is modular and generic. Dashboards are updated automatically based on the configuration files that can be edited after the first deployment with an idempotent logic.
- **Usability:** BABEL is developed in a way that makes it easy to use, thanks to a modular architecture that respects the best practices. Testers can use generic APIs and automatic deployment tools in different contexts.
- **Cloud native:** Most of Big Data systems were designed to be cloud native. For this reason, BABEL takes into consideration the cloud requirements thanks to its scalable and distributed technologies. It can benchmark any Big Data architecture in order to tune or compare it with other alternatives, either on the cloud or on premise.

8.2 Performance evaluation

This section aims to evaluate BABEL integration with the cash management platform, as well as its performance. This use case helps to validate the genericity and ease of use of our solution, as we integrate the tool to one of the most complex Big Data architectures (Kappa).

For BABEL performance evaluation, and especially for the data generation module, we compare the same test results with one of the most used Benchmark tools in the Big Data world: HiBench and TeraGen.

But first, to highlight the end-to-end monitoring feature, we start by comparing the different consumers' and producers' overhead to the system without any benchmark integration.

8.2.1 Benchmark overhead and processing time

BABEL is developed in a way that minimizes its overhead, by using asynchronous operations in the benchmark integration layer, when interacting with the benchmark core.

In order to validate and measure the overhead, different tests are designed and implemented. Every operation is sent to the Benchmark Store, along with its timestamp, in real-time.

For the test purpose, a Hadoop Cluster is deployed, including 10 Unix workers with 512GB of RAM, 64 cores and 10 SSD disks for each node. At this stage, a basic Big Data architecture is used including Apache Hadoop and Apache Spark. Producers

Table 1 Comparative matrix between producers (unit: second)

Workload	HiBench	TeraGen	BABEL Producer
500 GB	406.05	413.05	405.63
1 TB	810.58	831.38	811.75
100 TB	4081.76	4188.62	4078.81

Table 2 Producer overhead with benchmark core integration (unit: second)

Workload	Producer without core integration	Producer with core integration	Overhead (%)
500 GB	405.63	405.75	0.0305
1 TB	811.75	812.04	0.0355
100 TB	4078.81	4080.31	0.0367

inject generated data into HDFS, then a Spark batch job consumes it, executes a sort algorithm using each time a new consumer type.

Producers' performance We compare BABEL with two standard Hadoop benchmarks: TeraGen [47] and HiBench [10]. In Table 1, the results of three test scenarios are presented. The workloads consist in injecting 500 GB, 1 TB and 100 TB of data into HDFS, while keeping the same configuration: number of files, file format, parallelism level, basic SUT environment, etc. The values displayed below represent the average value obtained after executing the benchmark three consecutive times for every scenario.

During the BABEL and HiBench tests, we observe in the monitoring tools that the SUT is maximally stressed. This behavior is also confirmed by the execution times that are approximately the same. Comparatively, TeraGen is clearly less efficient.

After these tests, we can consider that the producer and data generation layers give acceptable performance results compared to their competitors and are able to stress the SUT by generating a lot of data via a distributed and scalable architecture.

Producers' overhead Once we validate the producers' performance, the goal now is to measure their overhead. The first test scenario is executed without the integration of BABEL. Then, in the second scenario, we re-execute the tests by integrating the producer with the BABEL core.

As presented in Table 2, the test results show a very light overhead with values between 0.03 and 0.04% thanks to the asynchronous calls between the producers and the benchmark store. We consider that the producer's overhead is close to zero, which is only relevant when dealing with very precise SUT benchmark tests.

Consumers' overhead Following the same approach, we integrated several BABEL consumers with a Spark sort job using different modes. For every data load (500 GB, 1 TB and 100 TB), we compute the processing time of the spark job, without any consumer integration, then with an agent-based, embedded and plugin consumers. Table 3 shows the obtained results and the overhead of each consumer type compared to the execution time of the spark job without any consumer integration.

As expected, the lowest overhead belongs to the agent mode with values between 0.019 and 0.023%. This mode is based on independent processes deployed in the same

Table 3 Consumers overhead (unit: second)

Workload	Without consumers	Agent consumer		Embedded consumer		Plugin consumer	
		Value	Overhead (%)	Value	Overhead (%)	Value	Overhead (%)
500 GB	81.62	81.64	0.021	81.66	0.049	81.65	0.038
1 TB	159.13	159.16	0.023	159.21	0.052	159.19	0.041
100 TB	8050.43	8051.96	0.019	8054.70	0.053	8053.57	0.039

Table 4 Performance evaluation when varying Kafka replication factors

Kafka replication factor	Average throughput (events per second)	Average latency (ms)
2	1,432,110	642
3	1,125,180	813
4	734,110	1045

SUT nodes with an indirect impact. We also note that the embedded mode is the most intrusive, with values between 0.049 and 0.053%.

To conclude, even though all the overhead values are acceptable, we recommend the usage of the consumer agent mode when possible.

8.2.2 Benchmarking metrics and results for the cash management context

As explained in the demonstration section, we focus on varying the consistency level of each layer:

- Producers: by testing different values of the acknowledgements “acks”, which represent the received write-acknowledgment count required from Kafka before the producer’s write request is deemed complete.
- Kafka: by changing the replication factor of the used topics.
- HBase: by changing the replication factor of HDFS blocks as HBase is using it for data persistence.
- Spark: by defining different checkpoints periods which persist the analytic metadata for recovery. The checkpoint mainly contains : the Kafka offsets of consumed topics, the analytic states and outputs metadata.

The main measured metrics in this section are: throughput and latency per layer.

First layer (Apache Kafka) consistency tests The first set of tests stressed the first layer of the SUT which is Apache Kafka. In the beginning, only the replication factor was changed to respectively be two, three and four replicas.

The results of these first tests are presented in Table 4. As expected, we observe that the Kafka replication factor is proportional to the injection latency and inversely proportional to the throughput. But, the goal of these tests is to adjust the consistency level in order to respect the performance need, which is to have a latency of less than one second, according to the SLAs. For these reasons, we opted for a Kafka replication factor of three that represents a throughput average of 1,125,180 events/s, a latency around 813 ms and still guarantees an acceptable consistency level.

Once the replication factor is defined, the tests will focus now on the Kafka producers’ consistency by varying the “ack” parameter to cover respectively one, two and three replicas.

Table 5 represents the results of this set of tests. In this level, we kept two acks in the producer configuration, which guarantees that the modification is propagated to two Kafka replicas before write commit.

Second layer (Spark Streaming) consistency tests Spark Streaming jobs were executed with different checkpoint periods: 1 s, 5 s and 10 s. The checkpoint is persisted in the

Table 5 Performance evaluation when varying producers acknowledgments

Producer ACKs	Average throughput (events per second)	Average latency (ms)
1	1,422,130	653
2	1,125,180	813
3	721,150	1098

Table 6 Performance evaluation when varying spark checkpoint interval

Spark checkpoint interval	Average throughput (events per second)	Average latency (ms)
1	325,120	2321
5	851,930	1053
10	1,128,710	751

Table 7 Performance evaluation when varying HBase and HDFS replication factors

HDFS replication factor	Average throughput (events per second)	Average latency (ms)
2	881,530	1022
3	603,310	1213
4	429,710	1519

same HDFS as HBase. All the checkpoints are incremental by updating existing states and creating the new ones with a micro-batch value equal to one second. Table 6 presents the average of the Spark job throughput and latency for each checkpoint interval value. In this case, checkpoints with one second frequency were very expensive in terms of latency and don't respect the service level agreements. On the other hand, 10 s of checkpoint frequency is not very consistent and can cause integrity issues. Finally, the value of 5 s is a good compromise between performance and consistency.

Third layer (Apache HBase) consistency tests The same consistency tests were done for HBase by varying the HDFS replication factor (two, three and four replicas), which represents the persistence layer of Apache HBase. When rows are injected into HBase, in the background, transactional files (WAL files) and data files (HFiles) are replicated in HDFS via synchronous writes. Table 7 presents the results of the different tests and confirm the limits of HBase when the HDFS replication factor is high. For this reason, only 3 replicas are configured with an average of 603,310 write operations per second and a latency of 1213 ms.

9 Conclusion

Using Big Data solutions for financial applications presents a huge challenge for companies nowadays. These systems need to face several constraints: storing the huge amount of continuously flowing data, processing historical and incoming data,

optimizing the use of high performance hardware, and running different types of jobs concurrently on the same platforms [5]. All this, while dealing with the precision and consistency needed by financial institutions and clients. To be able to estimate the efficiency of the developed solution, a sophisticated benchmarking platform needs to be used. We propose in this context BABEL, a scalable and generic benchmarking platform, that comes with a negligible overhead, as it runs in parallel with the evaluated system.

We show how to apply BABEL on a cash management application and analyze the obtained performance results of this platform. That said, BABEL can be applied to other use cases and domains thanks to its genericity and independence from the technologies and business requirements.

In order to insure an effective communication around our solution, we chose to share its code publicly in Github, along with a user guide documentation. We also are in the process of publishing its results and implementation details in various research and industry publications.

Using an adequate benchmarking platform can be useful for data engineers, to choose the right architecture and technology stack for a certain need. In fact, as stated by Goes [6], one of the biggest sources of confusion when implementing a Big Data system is the highly diversified number of solutions offered for the developers. This problem, coupled with the difficulty to conduct these types of projects that require heterogeneous expertise, trigger the need for a standardized design methodology for Big Data teams, that takes into consideration data governance, agility and collaboration.

References

1. Yu S, Guo S (2016) Big data concepts, theories, and applications. Springer, Berlin. <https://doi.org/10.1007/978-3-319-27763-9>
2. Han R, Xiaoyi L, Jiangtao X (2014) On big data benchmarking. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 8807, No. 1, p 3. https://doi.org/10.1007/978-3-319-13021-7_1
3. Zhang P, Shi X, Khan SU (2018) QuantCloud: enabling big data complex event processing for quantitative finance through a data-driven execution. IEEE Trans Big Data 5(4):564. <https://doi.org/10.1109/tbdata.2018.2847629>
4. QYResearch (2019) Payments market size, share, trends, growth and forecast report 2025—valuates reports. Tech. rep., QYResearch
5. Tian X, Han R, Wang L, Lu G, Zhan J (2015) Latency critical big data computing in finance. J Finance Data Sci 1(1):33. <https://doi.org/10.1016/j.jfds.2015.07.002>
6. Goes P (2014) Big data and IS research. MIS Q 38(3):III–VIII
7. Brewer E (2012) Pushing the cap: strategies for consistency and availability. Computer 45(2):23–29. <https://doi.org/10.1109/MC.2012.37>
8. Poggi N, Carrera D, Call A et al (2014) ALOJA: a systematic study of Hadoop deployment variables to enable automated characterization of cost-effectiveness. In: Proceedings—2014 IEEE international conference on big data, IEEE big data 2014, pp 905–913
9. Kim K, Jeon K, Han H, Kim SG, Jung H, Yeom HY (2008) MRBench: a benchmark for map-reduce framework. In: Proceedings of the international conference on parallel and distributed systems—ICPADS, pp 11–18
10. Huang S, Huang J, Dai J (2010) The HiBench benchmark suite: characterization of the MapReduce-based data analysis. In: 2010 IEEE 26th International conference on data engineering workshops (ICDEW 2010), pp 41–51

11. Sangroya A, Serrano D, Bouchenak S (2013) MRBS: towards dependability benchmarking for Hadoop MapReduce. In: Lecture notes in computer science (LNCS), vol 7640, pp 3–12
12. Nambiar R (2014) A standard for benchmarking big data systems. In: Proceedings—2014 IEEE international conference on big data, IEEE big data 2014, pp 18–20
13. Ouaknine K, Carey M, Kirkpatrick S (2015) The Pig Mix benchmark on Pig, MapReduce, and HPC systems. In: Proceedings—2015 IEEE international Congress on big data, BigData Congress 2015, pp 643–648
14. Li M, Tan J, Wang Y, Zhang L, Salapura V (2015) SparkBench: a comprehensive benchmarking suite for in memory data analytic platform spark. In: Proceedings of the 12th ACM international conference on computing frontiers—CF '15, pp 1–8
15. Ghazal A, Rabl T, Hu M, Raab F, Poess M, Crolotte A, Jacobsen HA (2013) BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD international conference on management of data ACM, New York, SIGMOD '13, pp 1197–1208
16. Yao BB, Özsü MT, Khandelwal N (2004) XBench benchmark and performance testing of XML DBMSs. In: Proceedings—international conference on data engineering, vol 20, p 621
17. Gu L, Zhou M, Zhang Z, Shan MC, Zhou A, Winslett M (2015) Chronos: an elastic parallel framework for stream benchmark generation and simulation. In: Proceedings—international conference on data engineering 2015-May, p 101
18. Lu R, Wu G, Xie B, Hu J (2014) StreamBench: towards benchmarking modern distributed stream computing. In: 2014 IEEE/ACM 7th International conference on utility and cloud computing, pp 69–78
19. Capotă M, Hegeman T, Iosup A, Prat-Pérez A, Erling O, Boncz P (2015) Graphalytics: a big data benchmark for graph-processing platforms. In: GRADES'15 Melbourne, pp 7:1–7:6
20. Ngomo AN, Röder M (2016) HOBbit: holistic benchmarking of big linked data. *Ercim News* 105:46
21. Armstrong TG, Ponnepanti V, Borthakur D, Callaghan M (2013) LinkBench: a database benchmark based on the Facebook social graph. In: SIGMOD'13 New York, p 1185
22. Nai L, Xia Y, Tanase IG, Kim H, Lin CY (2015) GraphBIG: understanding graph computing in the context of industrial solutions. In: International conference for high performance computing, networking, storage and analysis, SC 15–20-November-2015
23. Luo C, Zhan J, Jia Z, Wang L, Lu G, Zhang L, Xu CZ, Sun N (2012) CloudRank-D: benchmarking and ranking cloud computing. *Front Comput Sci* 6(4):347
24. Ferrarons J, Adhana M, Colmenares C, Pietrowska S, Bentayeb F, Darmont J (2014) PRIMEBALL: a parallel processing framework benchmark for big data applications in the cloud. *Lect Notes Comput Sci* 8391 LNCS:109
25. Stockinger K, Bundi N, Heitz J, Breymann W (2019) Scalable architecture for big data financial analytics: user-defined functions versus SQL. *J Big Data* 6(1):1–24. <https://doi.org/10.1186/s40537-019-0209-0>
26. Peffers Ken, Tuunanen Tuure, Rothenberger Marcus A, Chatterjee Samir (2007) A design science research methodology for information systems research. *J Manag Inf Syst* 24(3):45
27. Chen GW, Wang MHL, Liu KFR, Chen TH (2010) Application of project cash management and control for infrastructure. *J Mar Sci Technol* 18(5):644
28. Han R, Jia Z, Gao W, Tian X, Wang L (2015) Benchmarking big data systems: state-of-the-art and future directions. *arXiv*, pp 1–9
29. Wang L, Zhan J, Luo C et al (2014) BigDataBench: a big data benchmark suite from internet services. In: Proceedings—international symposium on high-performance computer architecture, pp 488–499
30. Han R, John LK, Zhan J (2018) Benchmarking big data systems: a review. *IEEE Trans Serv Comput* 11(3):580
31. Cockcroft S, Russell M (2018) Big data opportunities for accounting and finance practice and research. *Aust Acc Rev* 28(3):323. <https://doi.org/10.1111/auar.12218>
32. Turner D, Schroeck M, Shockley R (2013) Analytics: the real-world use of big data in financial services. *IBM Global Business Services* 27
33. Kumar BS, Ravi V (2016) A survey of the applications of text mining in financial domain. *Knowl Based Syst* 114:128
34. Xing FZ, Cambria E, Welsch RE (2018) Natural language based financial forecasting: a survey. *Artif Intell Rev* 50(1):49. <https://doi.org/10.1007/s10462-017-9588-9>

35. Dong W, Liao S, Liang L (2016) Financial statement fraud detection using text mining: a systemic functional linguistics theory perspective. In: Pacific Asia conference on information systems, PACIS 2016—proceedings
36. Kraus M, Feuerriegel S (2017) Decision support from financial disclosures with deep neural networks and transfer learning. *Decis Support Syst* 104:38. <https://doi.org/10.1016/j.dss.2017.10.001>
37. Chen W, Lai K, Cai Y (2018) Topic generation for Chinese stocks: a cognitively motivated topic modeling method using social media data. *Quant Finance Econ* 2(2):279. <https://doi.org/10.3934/qfe.2018.2.279>
38. Koh HC, Low CK (2004) Going concern prediction using data mining techniques. *Manag Audit J* 19:462–476
39. Zhong RY, Newman ST, Huang GQ, Lan S (2016) Big data for supply chain management in the service and manufacturing sectors: challenges, opportunities, and future perspectives. *Comput Ind Eng* 101:572. <https://doi.org/10.1016/j.cie.2016.07.013>
40. Velivassaki TH, Athanasoulis P, Trakadas P (2019) UCASH: ATM cash management as a critical and data-intensive application. In: CLOSER 2019—proceedings of the 9th international conference on cloud computing and services science (Closier), p 642. <https://doi.org/10.5220/0007876606420647>
41. Li S, Yu H (2019) Big data and financial information analytics ecosystem: strengthening personal information under legal regulation. *Inf Syst e-Bus Manag* 18(4):891. <https://doi.org/10.1007/s10257-019-00404-z>
42. Ivanov T, Rabl T, Poess M, Queralto A, Poelman J, Poggi N, Buell J (2016) Big data benchmark compendium. In: Lecture notes in computer science (including subseries Lecture notes in artificial intelligence and lecture notes in bioinformatics), vol 9508, p 135
43. Baru C, Bhandarkar M, Nambiar R, Poess M, Rabl T (2014) Big data benchmarking. In: 5th International workshop, WBDB 2014, Potsdam, Germany
44. Cooper BF, Silberstein A, Tam E, Ramakrishnan R, Sears R (2010) Benchmarking cloud serving systems with YCSB. In: SoCC '10 Proceedings of the 1st ACM symposium on cloud computing, Indianapolis, pp 143–154
45. Persico V, Pescapé A, Picariello A, Sperlì G (2018) Benchmarking big data architectures for social networks data processing using public cloud platforms. *Future Gener Comput Syst* 89:98
46. Kreps J (2014) Questioning the lambda architecture, Online article, July, p 205
47. Mohapatra D (2013) Terasort using MapReduce. Tech. rep

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.