Computer Science • 24(3) 2023 https://doi.org/10.7494/csci.2023.24.3.4831

Rafal Mucha
Bartosz Balis
Costin Grigoras
Jacek Kitowski

DATABASE REPLICATION FOR DISCONNECTED OPERATIONS WITH QUASI REAL-TIME SYNCHRONIZATION

Abstract *Database replication is a way to improve system throughput or achieve high availability. In most cases, the use of an active-active replica architecture is efficient and easy to deploy. Such a system has CP properties (from the CAP theorem: consistency, availability, and network-partition tolerance). Creating an AP (available and partition-tolerant) system requires the use of multi-primary replication. Because of the many difficulties in its implementation, this approach is not widely used; however, ALICE's deployment of CCDB (experiment conditions and calibration database) needs to be an AP system in two locations. This necessity became the inspiration for examining the state-of-the-art methods in this field and testing the available solutions. The tests that were performed evaluated the performance of the chosen replication tools: Bucardo, and EDB Replication Server; these showed that the tested tools could be successfully used for the continuous synchronization of two independent database instances.*

Keywords multi-primary database replication, Bucardo, EDB Replication Server, PostgreSQL, PostgresBDR, TPC, CAP theorem, continuous synchronization

Citation Computer Science 24(3) 2023: 401–420

1. Introduction

Database replication is a widely adopted solution [27] for improving the availability, reliability, and throughput of databases. The well-known CAP theorem [16] states that, in distributed systems, one can achieve only two of the three desirable qualities: consistency (C), availability (A), and tolerance to network partitions (P). Consistency and network-partition tolerance (a CP system) can be achieved by blocking write operations when a partition occurs, albeit at the cost of availability. Much more challenging is to achieve an AP system where clients are allowed to write to possibly partitioned systems at the cost of eventual consistency, wherein the system may not be globally consistent at all times. Furthermore, replication should be fast and lightweight – especially in soft real-time environments. The benchmarking of databases is a mature topic. In particular, there are some benchmarks like TPC [18] or YCSB [20] that are industry standards. However, there are no benchmarks that are focused on multi-primary database replication. This article is concerned with the problem of multi-primary replication. Apart from presenting the problem and its solution, it examines and compares the selection of currently available tools for multi-primary database replication; these are built by using different mechanisms, architectures, and technologies. All of these variables impact their stability, speed, and offered features. To compare the tools in a real-case scenario and experimentally evaluate their properties, the deployment of CCDB [24] (the experiment condition and calibration database) in the ALICE experiment [12] at CERN was used. To provide the high availability of CCDB, two instances of it were deployed in two processing centers.

1.1. Motivation and problem statement

In the upcoming Run 3[1], which was scheduled to start in 2022, the ALICE experiment is going to change the data-filtering model. In previous runs, ALICE experiments have used FPGAs [23] for basic reconstruction and filtering events in "online" processing. When LHC was built, this was a sufficient method for handling such high data rates. Almost a decade later, however, the current equipment is ready to process raw data streams without filtering [22]. A farm of more than 500 machines was created and backed with GPU accelerators; these will reconstruct, filter, and compress the data stream from detectors. During the upgrade, a new database system was developed to store the condition and calibration data of the experimental devices (CCDB [24]). This data is used at every level of data processing – either in the "online" processing or, later, in the "offline" grid processing of the results. The data in CCDB has millisecond granularity, so it is crucial to propagate the changes throughout the machine farm

[1]The LHC work is organized into "runs." LHC is used by all experiments (each one needs a different beam setup, so each experiment needs to reserve time and specify the beam parameters). The run takes a few years; after this, there is a planned break (a "long shutdown") when each experiment has access to detectors and upgrades or modifies its hardware for a new run. For now (March 2022), it is the second long shutdown.

in time and efficiently so that the "online" reconstruction nodes have actual data to work correctly. A way to achieve this is presented in Section 2. Being a critical part of the data-processing system, CCDB must be deployed to achieve high availability (HA), resilience, and network-partition tolerance. To enable HA, both the "online" and "offline" processing nodes have their own instances of CCDB. Both instances must be writable and should synchronize in quasi-real-time as long as there is a connection between them. An important requirement is to prevent data loss in the case of *reconciliation conflicts*; i.e., insert-insert conflicts (where rows with the same key are simultaneously inserted into different database instances). Previous tests have shown that the best choice for storing the CCDB database is PostgreSQL [36], as it has a unique feature that allows one to create an index for time periods.

1.2. Objectives

The main objective of the research work that is described in this article is to design and experimentally evaluate a solution for the quasi-real-time synchronization of independent database instances. The solution must satisfy the highly demanding data rates of the ALICE experiment: the condition data will be written with a frequency of 100 Hz (per measured parameter). The system must be highly available (A) and tolerant to network partitions (P). Any inconsistencies that occur during the partitioning must be reconciled after the connection has been restored. A particular research question of interest is this: "Could current multi-primary replication solutions be used in the implementation of a highly available and soft real-time system?" Due to time and resource limitations, the experiments were conducted for a PostgreSQL-based solution; however, the solutions were conceptually designed in a broader technology-agnostic scope. Other more precise questions of interest are as follows: "What is the overhead that is caused by these tools?" "How fast is the reconciliation process?" "How stable is the solution?" Answers to these questions should allow one to choose the best option in the given use case. Answering other questions ("How do these tools work?" What are their architectures, what databases mechanisms do they use, and how?" should allow for a better understanding of the limitations or potential problems and a more conscious choice of tools.

1.3. Research methodology

The methodology that was used to achieve the objectives consisted of two parts. The first part was an evaluation of the tools for multi-primary replication using a simplified case-study approach. The second part was building a solution for the multi-primary PostgreSQL database connection as well as its experimental evaluation and performance tuning.

1.3.1. Evaluation of database-replication tools

One of the parts of this work was to gather information about the available tools for multi-primary replication. Their evaluation took multiple criteria into account,

such as whether the tools were still maintained, what the supported conflict-handling strategies were, how they operated, or what the deployment architecture was.

Some of the tools are built into DBMS systems, while others are standalone. The results of the evaluation consist of a short description of each tool that covers the points that were mentioned earlier.

1.3.2. Database-replication performance

One of the main objects of concern is the performance of the database when working with two primary replication configurations. In many cases, the database could fit into the memory so that the read performance would not be affected by the replication and an evaluation of the reading performance would not be needed. The plan for the experimental evaluation consisted of the following steps:

- finding maximal insertion rate of database without replication tools;
- finding overhead that is caused by replication tools to PostgreSQL with active replication and finding maximal stable insert rate when replication is active;
- measuring reconciliation time (how fast replicas reconcile after longer disconnection) – additionally, it should be verified whether databases are blocked by reconciliation process.

To collect the experimental data, the databases were monitored during the tests. Among the monitored parameters were the number of operations per second (fetch, insert, read) and the number of inserted rows per second. There were some improvements in the configurations following the documentation (within the scope that was possible); however, the configuration could probably have been improved due to the complexity of these tools. For the reproducibility, the crucial changes in the configuration of PostgreSQL and the the used tools are given. During the tests, metrics like the number of fetched, inserted, and returned rows from the database, the metrics from the replication tools that showed the number of sent rows, and the reconciling time were measured. The tests were mainly carried out in Cyfronet by using 12 machines that were part of the Zeus supercomputer [11]. They were multi-core nodes (12 physical cores with Hyper-Threading) with 24 GB RAM (enough to fit a database in the memory) and a fast connection (Infiniband with an IPoIB network). The Infiniband network was used to mimic the conditions in the target cluster.

1.4. Novelty of approach

Database benchmarking is a mature topic; for example, there are some well-known benchmarks such as TPC [18] that are industry standards. However, there is a lack of tests on multi-primary replication solutions. This work proposes simple yet comprehensive scenarios for testing multi-primary replication. Following these scenarios, two tools were tested: Bucardo [6], and EDB Replication Server [3]. Another important contribution was to review and describe the selected tools that support multi-primary replication.

2. ALICE condition and calibration database system architecture

This section presents the deployment of CCDB [24] – experimental conditions and calibration database in the ALICE [12] experiment at CERN. This is an essential part of the newly deployed Run-3 software for "online" processing because it stores and serves data that is a critical part of the whole experiment. Consequently, this database must be highly available and resilient.

2.1. CCDB

CCDB is a database that stores data about experimental conditions; this data is usually stored in the ROOT format [17]. Binary files are stored in local disks and uploaded for resilient storage to selected grid storage services. Metadata like the device path, validity, and path to file copies is held in the PostgreSQL database. The deployment of CCDB is shown in Figure 1.

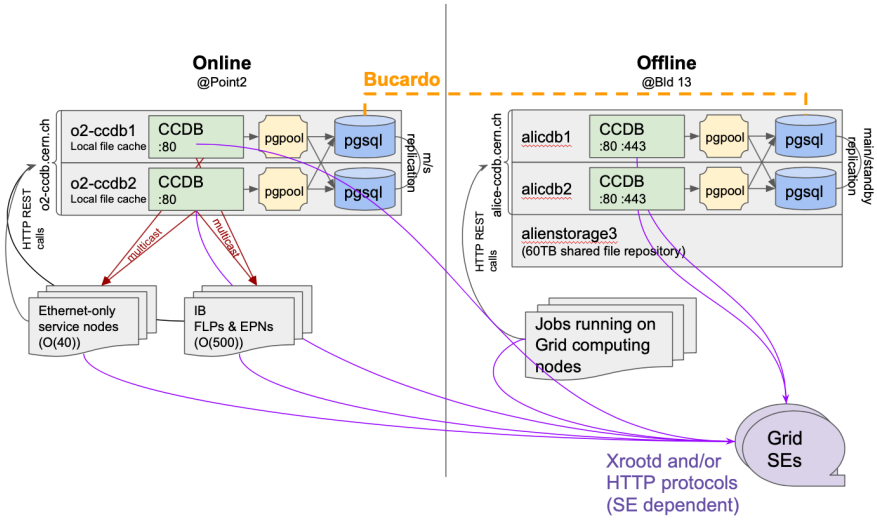

Figure 1. CCDB deployment

The first place where a CCDB instance runs is the "online" side – the processing center that is directly connected to the experiment detectors where the synchronous reconstruction runs. This processes the data stream from the detector, runs the first reconstruction, and filters the data. This processing center consists of about 540 machines, which will run the software that uses the data from CCDB. Another difficulty is the data's update rate, which can reach about 100 Hz per parameter. To efficiently handle the data distribution, CCDB supports work in proxy mode. The data is distributed by a UDP multicast to all proxy instances immediately after they have been uploaded to the main instances of CCDB in the cluster (which are

backed by PostgreSQL). The second CCDB instance runs on the "offline" side. This instance serves the data for the "offline" grid processes that are running in many HPC and HTC centers around the world. The grid processes could upload new data; for example, new detector calibration. "Online" and "offline" sides are synchronized by PostgreSQL databases, which store metadata about condition files (binaries are stored locally and in CERN's grid storage).

2.2. Architecture extensions

To increase the resilience and load balancing, the PostgreSQL database deployment was extended as follows:

- Primary-replica replication – the database on each side is replicated by using the built-in streaming replication on the secondary node. In the event of damage to the primary node, the replica machine can be quickly promoted to the primary role with minimal data loss (missing only what could not be written during this transition period).
- Connection pooling – helps restrict the number of connections that are open to the database; however, it has more functionality. The chosen Pgpool-II [10] supports caching data, detecting the primary node in the streaming replication, and calculating replication delay (if this is greater than a set value, it redirects all selects to the primary node). Another feature is the detection of a primary node and the redirection of the insert operation to the primary.
- Multi-primary replication – provides continuous synchronization of the "online" and "offline" databases. After a connection between the two sides has been (re)established, the two databases should be reconciled gently. This is performed by Bucardo, which runs on the "online" primary node; this is an open-source tool that supports multi-primary replication for PostgreSQL. As shown in Section 5, it is robust enough to handle any expected write loads and successfully synchronize both instances.
- Monitoring – all components of the deployment architecture need constant monitoring. The metrics for PostgreSQL, Bucardo, and Pgpool-II are reported to MonALISA [30]; this allows for monitoring the health of the services and alerting the shifter crew in case of problems.

In summary, the proposed architecture should allow for continuous work even in the case of hardware or software failure. It should also enable high data throughput during a running experiment and later analysis. It should also continue working even in the case of disconnecting the "online" and "offline" sides. The architecture that is presented in Fig. 1 was designed by ALICE. The main contribution of this work to the presented architecture was selecting, deploying, and testing a solution to synchronize the "online" and "offline" instances as two primaries.

3. PostgreSQL multi-primary replication tools

PostgreSQL is a mature and popular database system. Over the years, numerous projects have been developed that have tried to tackle the problem of multi-primary replication. Due to the PostgreSQL architecture, two mechanisms have been used to detect changes in the database. The first consists of triggers. During initialization, the tool creates internal tables and triggers. Triggers are connected to each replicated table and save the indexes of any affected rows to its internal tables. Later, the tool reads the data from the internal tables and replicates the modifications. The second mechanism, which is utilized to detect changes, consists of write-ahead logger (WAL) plugins. A write-ahead logger is a mechanism that allows for persistent and consistent work without the need of dumping a whole database after each write transaction. Databases that use WAL can operate in the memory. The task of WAL is to persistently store the logs of the operations that are performed on the data. Writing such logs to a disk is a much lighter operation than writing down all of the modified data. From the perspective of a multi-primary replication tool, it could also be used as a source of data on modifications. Below, the three available tools for multi-primary replication that belong to the PostgreSQL ecosystem are described; each has some unique properties that have allowed it to survive on the market to this day.

3.1. Bucardo

Bucardo [7] is an open-source tool for logical replication; its available replication modes are primary-replica (MS) or multi-primary (MM). This offers the best support for PostgreSQL (MM and MS), but other databases can be used as copy targets. The project was started in 2002 by Jon Jensen and Greg Sabino Mullane in Backcountry [6]; it is still being developed and available under a BSD license [8] and is written in Perl5 [37]. For marking modifications to propagate, it uses triggers and tracing tables. Each trigger also sends a message to Bucardo's control process – communicating that a modification occurred, so the KID process should run. The tables can be grouped into *sync*. Then, *sync* is synchronized by a KID process. In general, Bucardo has a three-tier running architecture: Bucardo *master control process* (MCP), Bucardo *control processes* (CTL), and KID. MCP monitors and manages all of the other processes. CTL spawns and controls the KID processes. KID is a process that does the real replication work; it connects to each database to read tracing tables and check and resolve conflicts on primary keys. Then, it reads the actual data and tries to propagate the changes to other databases. During the insertion of data to these, exceptions can occur (such as violating unique constraints).

Custom codes can handle conflicts and exceptions automatically; these are functions that receive the object that describes the current state of the KID or CTL process and can try to resolve conflicts or the source of an exception. In general, such an attempt to reconcile scripts allows for a lot of elasticity in invoking compensation actions. Before using the default conflict strategy for the table, *conflict* custom codes that are associated with a given table are generally invoked. If they do not resolve the

conflicts, then the default strategy is used. As shown, Bucardo is a sophisticated tool; it has many options to configure multi-primary replication in a maximally automated and data loss-free way. However, the downside of Bucardo is that it is a single point of failure in the system.

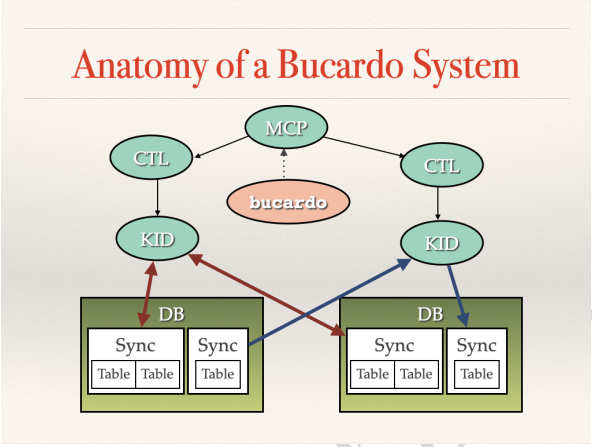

Figure 2. Bucardo running scheme, source [19]

3.2. EDB replication server

EDB xDB (cross-database) Replication Server [3] is another replication system. xDB Replication Server supports two different replication models: single-primary (primary-to-secondary) replication, and multi-primary replication. In the single-primary replication mode, it supports PostgreSQL, Advanced Server, Oracle, and Microsoft SQL Server. For multi-primary replication, xDB Replication Server supports configurations that consist of PostgreSQL database servers and EDB PostgreSQL advanced servers. An illustration of the multi-primary configuration is presented in Figure 3. For storing and propagating information about updates in databases, it uses *processes* that are spawned by each database instance and writes the data to Kafka [29] *streams*. ZooKeeper [26] manages the configuration. For custom reconciliation handlers, it uses functions that are stored in the database. After adding a function to the database, it must also be added to the configuration as a conflict resolver for a given table. In terms of the mechanisms that are used to detect changes, xDB can utilize both the trigger and write-ahead logger. The application has CLI and GUI.

In summary, EDB Replication Server is mature and even more sophisticated than Bucardo. It supports both WAL and trigger-based replication, allows adding custom codes, and has multiple default strategies to resolve conflicts. The documentation is of a good quality. The only downside is that one needs sudo to install the EDB replication server on Linux systems.

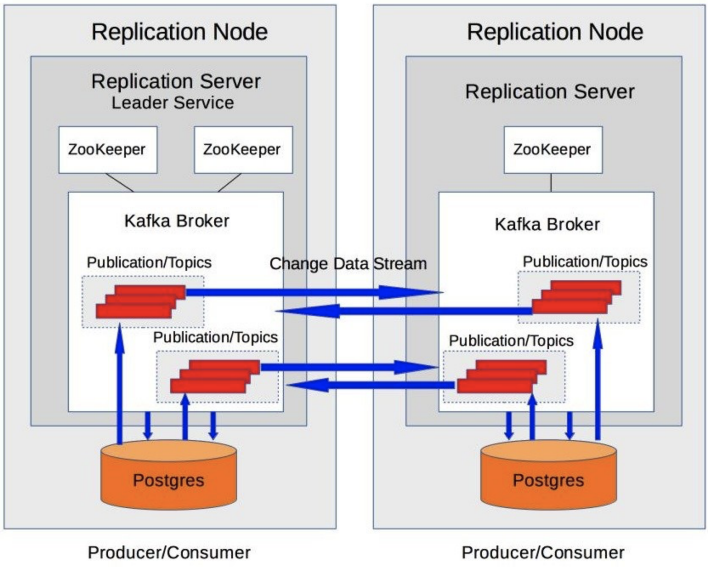

Figure 3. Running scheme of EDB Replication Server for multi-primary replication, source [3]

3.3. PostgresBDR

Generally, there are two tools under the BDR name: BDR [2] Version 2 (which is open-source but no longer maintained), and Version 3 (developed and offered by 2ndquadrant [4]), which is part of EDB [9]. It offers multi-primary logical replication. It is not an external application like Bucardo or xDB but is a PostgreSQL plugin. Thanks to being a plugin, it has access to a write-ahead logger and uses it to propagate information about modifications to other databases. Like Bucardo, it has two different modes for handling exceptions that occur in multi-primary mode. It can be predefined like last-write-win or by a user's custom code. Due to using such an architecture, this replication is lighter (it does not require additional writes on a disk like in the case of trigger-based tools) and faster (as it can propagate new data just after receiving it without any delay to other databases).

4. Related works

The performance of popular databases was measured and presented in [14] by comparing the work parameters of four DBMSes: PostgreSQL [36], MongoDB [15], Redis [21], and InfluxDB [34]. The test case in the paper was environmental monitoring, early warning, and decision-support systems (which are unique and difficult to compare with our use case). Furthermore, it does not evaluate the performance of replicated

databases (which is a must for such systems) if the network and the amount of collected information are to grow.

Kolonko's thesis [28] contained a ranking list of the most popular database systems and evaluated about 15 of the most popular papers that examined database performance. On this basis, he proposed the most common subset of benchmarks; this can be beneficial for preparing test scenarios that will be the most representative. It contained discussions of other papers that were evaluated, including actively pointing out any gaps or faults in the methodology or justifications of the databases that were used for the tests. The testing part of the thesis contained two databases: Oracle, and MongoDB. Unfortunately, it did not compare the performance of replicated databases.

Another group of articles about database benchmarking are those that describe and use TPC. TPC is a group of market-standard benchmarks for testing database performance. Among them are those that test OLTP (such as TPC-C, which isdescribed in detail here [25]). [18] contained a comparison of TPC-C and TPC-E – a new complementary benchmark to TPC-C that was introduced in 2007. In general, these benchmarks simulate some practical use cases. They are available as specifications, not as ready programs – so, they must be implemented by each tester. It improves portability, but it makes tests difficult and expensive [33]. Because of this (and because it does not provide an exact scenario for testing database replication), these benchmarks are not very useful in this work.

Cooper et al. [20] presented Yahoo! Cloud Serving Benchmark (YCSB). This is a benchmark that was designed for testing NoSQL databases like BigTable, PNUTS, Cassandra, or HBase rather than typical SQL databases. However, these tests contain sharded MySQL. Later, YCSB evolved, and some forks have appeared [35]. The authors of an article are testing different aspects that were missed by the original framework, focusing instead on key value stores. In summary, YCSB benchmarks are easier to use and closer to those that are tested in this work than TPC-like benchmarks; however, they do not measure the reconciliation time in case of disconnection.

Another example of benchmarking is presented in [13]. The study demonstrated a synthetic benchmark created by Facebook with the aim of comparing MySQL with HBase. The benchmark was designed to simulate production traffic from Facebook while avoiding the use of real traffic. However, despite such efforts, there is still no standard benchmark that can reliably and clearly compare different databases.

Moiz et al. [31] discussed the available tools for database replication. This article discussed open-source and commercial ones; however, it has rather limited descriptions and does not contain any performance check.

Another important work is [32]; it described a problem that was similar to ALICE that occurs during the creation system for ITER. The main problem was the implementation of multi-primary replication. This paper discussed the replication tools for PostgreSQL and for some other replication systems; however, it again lacks

a comparison of performance tests. The paper concluded that the best option for these was the PostgresBDR database.

5. Multi-primary replication performance

This section presents the results of the performance tests that compared Bucardo and xDB. CCDB was used to create the test scenarios. The first was a measurement of the differences in the PostgreSQL configuration, and the next test showed the overheads that were caused by adding tools to a database. The third test scenario contained measurements that ran with a limit-close insert rate for a longer period of time. In the final test scenario, the replica reconciliation time was measured. These tests together should allow us to determine the impact that is caused by using the evaluated tools, what their limitations are, and which tool is better.

5.1. Test environment

The test was run in the isolated part of the Zeus HPC cluster of the Cyfronet Computing Center. The machines had the following specifications:

- Processors: Intel(R) Xeon(R) CPU X5650 @ 2.67 GHz * 2;
- RAM: 24 GB;
- Disk: HDD GJ0120CAGSP HP 120-GB;
- Infiniband card: Mellanox Technologies MT26438 [ConnectX VPI PCIe 2.0 5GT/s – IB QDR / 10GigE Virtualization+].

The machines were connected to the Infiniband network (with IPoIB on top).

5.2. System under testing

The system under testing consisted of two instances of PostgreSQL 13.2. Their configuration was modified (e.g., *synchronous commit* was turned off) to get higher insert performance at the cost of the possibility of losing a small amount of data in the case of a power-down. When using the default settings, the insert rate was unsatisfactory. If the default configuration was used, the tests would be limited by the performance of the disk. This is discussed in detail below.

In the test scenarios, the two databases were synchronized using Bucardo and xDB Replication Server. Both tools used triggers that saved modified indexes in the database. The advantage of xDB is that it does not create additional databases and users (as Bucardo does); instead, the instructions describe how to create a user with sufficient rights in the system.

For the insert tests, benchmark programs were prepared that used CCDB's Java code. The logs were sent to the MonALISA service (whose instance was deployed on Cyfronet). An overview of the system is presented in Figure 4.

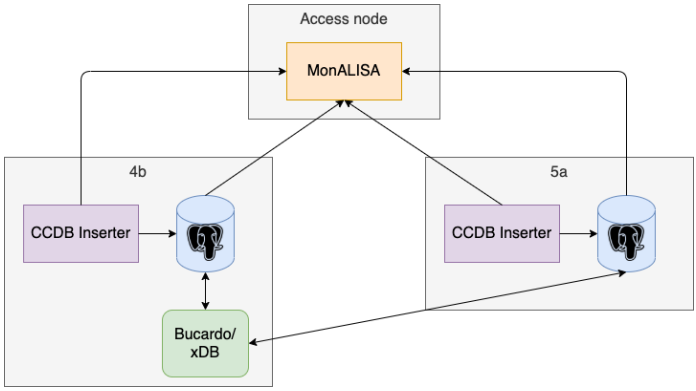

Figure 4. Architecture of system to test replication. Inserters that are based on CCDB code insert objects to PostgeSQL. Insert rate is reported to MonALISA instance. Database instances are synchronized using Bucardo or xDB, whose instance is on Node *4b*. MonALISA also gathers metrics from PostgreSQL instances.

5.2.1. PostgreSQL

Crucial for write performance was the setting of the write-ahead logger. The following values were experimentally determined as a suitable trade-off between resilience and performance (*synchronous_commit = off, wal_writer_delay = 100 ms*). In general, the performance difference between the synchronous and asynchronous commits was the greatest because of slow disks. While the maximum insert rate that databases can handle is about 80 Hz with synchronous commit, turning this off allows one to achieve an insert rate of 1.5–2.5 kHz.

5.2.2. Bucardo

For Bucardo, one of the parameters that could affect performance is persistent KID and CTL (Bucardo sync options: *kidsalive=false stayalive=false*). After this parameter was disabled, the database was more stable during long-running tests.

5.2.3. EDB replication server

The EDB replication server's configuration was close to default; it did not contain conflict handling codes, was configured to use triggers, and was scheduled to run every second (the shortest possible interval). An improvement was to increase the limit of the working memory to 16 GB (JVM options: *-Xms4096m -Xmx16384m*).

5.3. Measured metrics

The metrics were collected using the MonALISA monitoring system. Logs from inserter programs were sent every second and were aggregated into two-minute windows, while the system metrics were gathered every 30 seconds.

- Number of insert transactions served in the database – write operations were the most critical; since the CCDB's database could fit in the memory, read operations were not critical and should not have influenced the database's performance.
- Reconcile time (in selected tests) – the time it takes a tool to reconcile two databases after a longer disconnection.
- Number of fetched/returned/inserted/deleted rows – to check which tool generates a higher OPS load for the database.

5.4. Tests

5.4.1. PostgreSQL configuration

The first test measured the impact of changing the *synchronous commit* option; it was changed to *off* due to the limited insert performance when using the default configuration. In numbers, it allows one to reach about a 1750 Hz insert rate against about 80 Hz when using the default settings. Since turning off the *synchronous commit* option is safe for consistency and can only lead to small data losses in the event of power outages, this option is acceptable as a production configuration in the ALICE experiment.

5.4.2. Insert to database

This test shows the overhead that was caused by the triggers that were added by the replication tools. The compared configurations were a vanilla PostgreSQL, PostgreSQL with initialized Bucardo, and PostgreSQL with initialized xDB. The time to insert N objects was compared (where N was 10k, 100k, and 1M), and each test was repeated three times. Initially, the database was empty. The results are presented in Figure 5.

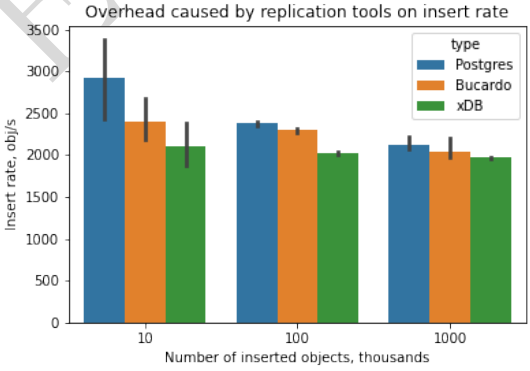

Figure 5. Average insert rate depending on number of inserted objects and replication tool. Each test was run three times. During test, replication was not active to show overhead caused by triggers added by replication tools.

As can be seen, an overhead is visible; however, it was not significant. Bucardo seemed to perform better than xDB for all of the tested cases.

5.4.3. Reconcile after disconnection

The test evaluated the speed at which a given tool reconciled the two databases. At first, the database contained ten million objects.

The test scenario was as follows:

- turn off replication tool;
- run *VACUUM* operation[2] and then insert one million objects to both instances (inserters run first *VACUUM* operation to prevent it from running later);
- turn on inserters to both databases with small insert rate (about 50 Hz);
- turn on replication tool – monitor total time for reconciling and impact on database performance.

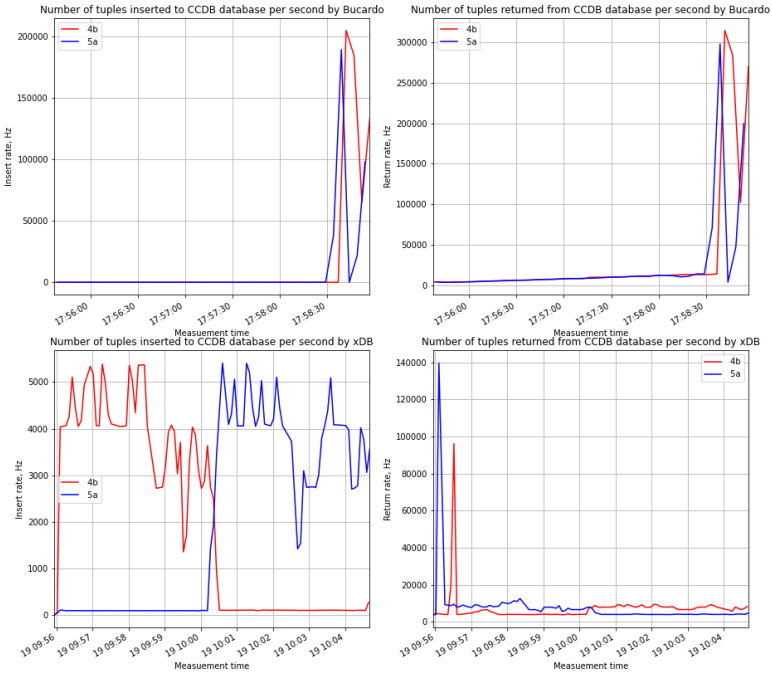

Figure 6. Rate of insert (left) and return (right) operations in database while doing reconciliation using Bucardo (upper) and xDB (down). Red lines are results of Node *4b* (which runs Bucardo/xDB), while blue lines are results of Node *5a*. As can be seen, tools had different load characteristics.

[2]VACUUM is a garbage-collection operation. "VACUUM causes a substantial increase in I/O traffic, which might cause poor performance for other active sessions" following the PostgreSQL documentation [1]. Then, it was executed manually in order to reduce the possibility that this operation would be fired during the tests.

The results were as follows: Bucardo reconciled two databases with 1M new objects in each in about 200 seconds, while the same operation took about 500 seconds for xDB. As one can see, the performance of Bucardo was significantly better (about 2.5x times) than that of xDB.

Figure 6 presents the load of the database system during the test. The behavior of both tools was quite different; xDB continuously synchronized small packets of rows with each other, while Bucardo read the indexes of the rows to first synchronize and then exchange all of the data at the maximum speed. On the one hand, it was a much more efficient method, but on the other hand, the maximal load that was generated by the tool could not be controlled nor restricted to the database throughput (e.g., 50%).

5.4.4. Reconcile after disconnection – different numbers of objects to reconcile

The next test was similar to the previous one. The reconciliation times vs. the number of objects to reconcile (10k, 100k, 1kk) were measured. The results are presented in Figures 7 and 8.

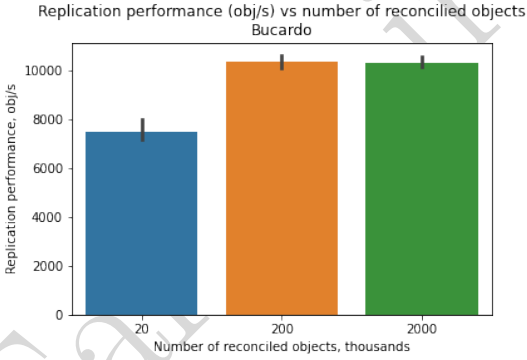

Figure 7. Bucardo – reconciliation performance vs. number of reconciled objects

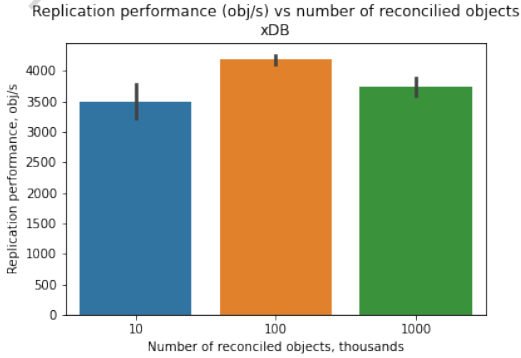

Figure 8. xDB – reconciliation performance vs. number of reconciled objects

As can be seen, the best results were for 200k objects in both cases. In conclusion, it was more efficient to run the reconciliation in real-time for a greater amount of information less frequently than it was for lesser information in smaller batches. Additionally, the tools suffered a little in their performance during the replication of too many objects; the optimal value was between 2k and 20k objects.

5.4.5. Continuous work with higher near-limit load

The objective of this test was to find the maximum possible insert rate of continuously synchronized databases. For Bucardo, the experimentally found value was 500 Hz (two inserters – each with a 250 Hz insert rate). When the insert rate was about 600–700 Hz, the tool became unstable: after a few minutes of work, the replication was locked (apparently because of going into an infinite-retry loop). During the locking, there were an increasing number of fetched and returned rows from the database (blocking normal work) such that the inserter was affected – the insert rate dropped below 50 Hz. It turns out that xDB was able to run the system smoothly with a 1000 Hz insert rate (2 * 500 Hz); this was an impressive result. On the other hand, Bucardo started to throttle (lock) several minutes after the start when faced with such a load. Insert rates that were greater than 500 Hz were not stable with xDB, and the insert rate was throttled.

5.5. Conclusion

As the experiment showed, Bucardo was lighter and reconciled the databases faster; on the other hand, xDB was much better in continuous synchronization. Judging by the current evaluation, however, Bucardo could be a better choice. Finally, Bucardo was also used to reconcile the CCDB databases in ALICE. The choice will be reevaluated after several months of work in the production environment – especially when it will be faced with a real load.

6. Summary and future work

Multi-primary replication has been shown to be a challenging and important problem. By its nature, there is no solution that will work correctly out-of-the-box despite the fact that the problem has been known for many years. The minimal configuration that must be provided is a conflict-resolution strategy that allows us to manage conflicts in order to keep the database consistent.

The main contribution of this work is the testing of multi-primary replication tools. In the literature, no standard test method can be found to benchmark multi-primary replication. Consequently, test scenarios were created using the CCDB database as a system under test that covered different aspects of multi-primary replication. Thanks to this method, CCDB could be easily adopted to a different database to test another tool. The created test scenarios covered several important aspects: (1) the overhead that was caused by using the replication tool; (2) the batch replication of multiple objects (simulating a longer disconnection); and (3) checking whether

the tool could steadily handle a high insert rate during continuous synchronization (a mode of operation that is characteristic of the quasi-real-time synchronization of databases). All of these showed overheads, which could be expected when applying a multi-primary replication tool to a system with continuous synchronization.

Possible directions to continue the research that is presented in this work are as follows:

- Test extensions can be performed in two possible ways: the first is to extend existing test scenarios by reconciliation tests with conflicts and comparisons of other tools (the test may include a geographical distribution of nodes to increase the communication overhead), and the second is to prepare new (or expand existing) test scenarios, which allows for the testing of cloud services (such as Azure CosmosDB [5]).
- Another possible way to tackle the problem of benchmarking multi-primary replication can be done by writing benchmark scenarios that are similar to TPC or writing a generic benchmark engine for evaluating multi-primary replication. Later steps in this approach will be to adopt benchmarks for databases and run tests on them.

6.1. Summary

Summing up, the answer to the question that was stated in the objectives of this work is as follows: existing multi-primary replication tools can be applied to highly available soft-realtime systems. The main objective of this work – the design and experimental evaluation of a solution to synchronize two database instances – was successfully realized. In addition, it was experimentally proven that the proposed solution can handle a load that is ten-times higher than that which is required by ALICE's CCDB use case. Using the designed solution, tests were conducted that concluded that the overhead that is caused by multi-primary replication tools is significant but not disruptive.

This work also has a significant practical impact. The architecture that was proposed in Section 2 was successfully implemented and deployed as part of the ALICE O2 system; so, it will be critical in future ALICE runs (which will likely lead to new discoveries in physics).

Acknowledgements

RM, BB, and JK were partly supported by the Polish Ministry of Education and Science (Agreement Nr. 2022/WK/1) and by the funds of the Polish Ministry of Education and Science assigned to AGH University of Science and Technology.

References

[1] PostgresSQL - official documentation, https://www.postgresql.org/docs, Accessed Aug 17, 2021.

[2] BDR 1.0.7 Documentation, http://bdr-project.org/docs/stable/, Accessed March 30, 2021.

[3] EDB Replication Server (6.2) Official Documentation, Accessed March 30, 2021. https://www.enterprisedb.com/edb-docs/static/docs/eprs/6.2/EDB_Postgres_Replication_Server_Users_Guide_v6.2.pdf.

[4] 2ndquadrant - official page, https://www.2ndquadrant.com/en, Accessed May 25, 2021.

[5] Azure Cosmos DB – conflicts handling documentation, Accessed May 25, 2021. https://docs.microsoft.com/en-us/azure/cosmos-db/how-to-manage-conflicts.

[6] Backcountry company page, https://backcountry.com, Accessed May 25, 2021.

[7] Bucardo - official page, https://bucardo.org/Bucardo, Accessed May 25, 2021.

[8] Bucardo repository page, https://github.com/bucardo/bucardo, Accessed May 25, 2021.

[9] EnterpriseDB – official page, Accessed May 25, 2021. https://www.enterprisedb.com.

[10] Pgpool-II - official page, https://www.pgpool.net, Accessed May 25, 2021.

[11] Zeus supercomputer page in ACK Cyfronet, Accessed May 25, 2021. https://www.cyfronet.krakow.pl/13385,artykul,superkomputer_zeus.html.

[12] ALICE experiment page, https://alice-collaboration.web.cern.ch, Accessed October 22, 2020.

[13] Armstrong T.G., Ponnekanti V., Borthakur D., Callaghan M.: Linkbench: a database benchmark based on the facebook social graph. In: *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pp. 1185–1196, 2013.

[14] Balis B., Bubak M., Harezlak D., Nowakowski P., Pawlik M., Wilk B.: Towards an operational database for real-time environmental monitoring and early warning systems. In: *ICCS*, pp. 2250–2259, 2017.

[15] Boicea A., Radulescu F., Agapin L.I.: MongoDB vs Oracle–database comparison. In: *2012 third international conference on emerging intelligent data and web technologies*, pp. 330–335, IEEE, 2012.

[16] Brewer E.A.: Towards robust distributed systems. In: *PODC*, vol. 7, pp. 343477–343502, Portland, OR, 2000.

[17] Brun R., Rademakers F.: ROOT - an object oriented data analysis framework, *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 389(1-2), pp. 81–86, 1997.

[18] Chen S., Ailamaki A., Athanassoulis M., Gibbons P.B., Johnson R., Pandis I., Stoica R.: TPC-E vs. TPC-C: Characterizing the new TPC-E benchmark via an I/O comparison study, *ACM Sigmod Record*, vol. 39(3), pp. 5–10, 2011.

[19] Christensen D.: Bucardo presentation 1, https://bucardo.org/Bucardo/presentations/2015-Choosing-Logical-Replication.pdf, Accessed March 30, 2021.

[20] Cooper B.F., Silberstein A., Tam E., Ramakrishnan R., Sears R.: Benchmarking cloud serving systems with YCSB. In: *Proceedings of the 1st ACM symposium on Cloud computing*, pp. 143–154, 2010.

[21] Da Silva M.D., Tavares H.L.: *Redis Essentials*, Packt Publishing Ltd, 2015.

[22] Eulisse G., Konopka P., Krzewicki M., Richter M., Rohr D., Wenzel S.: Evolution of the ALICE Software Framework for Run 3. In: *EPJ Web of Conferences*, vol. 214, p. 05010, EDP Sciences, 2019.

[23] Grastveit G., Helstrup H., Lindenstruth V., Loizides C., Roehrich D., Skaali B., Steinbeck T., Stock R., Tilsner H., Ullaland K., *et al.*: FPGA co-processor for the ALICE high level trigger, *arXiv preprint physics/0306017*, 2003.

[24] Grigoras C.: CCDB Conditions DB for Run 3, Accessed May 25, 2021. https://docs.google.com/presentation/d/1RMIzqHL1JnDhwmqGj_yTmxqjNb54hNoJwvFIgtldR6g/edit#slide=id.g25765cf80e_0_3.

[25] Hsu W.W., Smith A.J., Young H.C.: Characteristics of production database workloads and the TPC benchmarks, *IBM Systems Journal*, vol. 40(3), pp. 781–802, 2001.

[26] Hunt P., Konar M., Junqueira F.P., Reed B.: ZooKeeper: Wait-free Coordination for Internet-scale Systems. In: *USENIX annual technical conference*, vol. 8, 2010.

[27] Kemme B., Jiménez-Peris R., Patiño-Martínez M.: Database replication, *Synthesis Lectures on Data Management*, vol. 5(1), pp. 1–153, 2010.

[28] Kolonko K.: Performance comparison of the most popular relational and non-relational database management systems, 2018.

[29] Kreps J., Narkhede N., Rao J., *et al.*: Kafka: A distributed messaging system for log processing. In: *Proceedings of the NetDB*, vol. 11, pp. 1–7, 2011.

[30] Legrand I., Cirstoiu C., Grigoras C., Voicu R., Toarta M., Dobre C., Newman H.: *MonALISA: An agent based, dynamic service system to monitor, control and optimize grid based applications*, CERN, 2005.

[31] Moiz S.A., Sailaja P., Venkataswamy G., Pal S.N.: Database replication: A survey of open source and commercial tools, *International Journal of Computer Applications*, vol. 13(6), pp. 1–8, 2011.

[32] Nakanishi H., Yamanaka K., Tokunaga S., Ozeki T., Homma Y., Ohtsu H., Ishii Y., Nakajima N., Yamamoto T., Emoto M., Ohsuna M., Ito T., Imazu S., Nonomura M., Yoshida M., Ogawa H., Maeno H., Aoyagi M., Yokota M., Inoue T., Nakamura O., Abe S., Urushidani S.: Design for the distributed data locator service for multi-site data repositories, *Fusion Engineering and Design*, vol. 165, p. 112197, 2021. doi: https://doi.org/10.1016/j.fusengdes.2020.112197.

[33] Nambiar R., Poess M.: Keeping the TPC relevant!, *Proceedings of the VLDB Endowment*, vol. 6(11), pp. 1186–1187, 2013.

[34] Naqvi S.N.Z., Yfantidou S., Zimányi E.: Time series databases and influxdb, *Studienarbeit, Université Libre de Bruxelles*, p. 12, 2017.

[35] Reniers V., Van Landuyt D., Rafique A., Joosen W.: On the state of nosql benchmarks. In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*, pp. 107–112, 2017.

[36] Stonebraker M., Rowe L.A.: The design of POSTGRES, *ACM Sigmod Record*, vol. 15(2), pp. 340–355, 1986.
[37] Wall L., *et al.*: The Perl programming language, 1994.

Affiliations

Rafal Mucha
AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, mucharafal44@gmail.com; CERN, Geneva, rafal.mucha@cern.ch

Bartosz Balis
AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, balis@agh.edu.pl

Costin Grigoras
CERN, Geneva, grigoras@cern.ch

Jacek Kitowski
AGH University of Science and Technology, al. Mickiewicza 30, 30-059 Krakow, kito@agh.edu.pl

Received: 06.05.2022
Revised: 07.07.2022
Accepted: 07.07.2022