**University of Waterloo**

Faculty of Mathematics

**Analysis of the Demand and Solution of**

**Parallel Recovery in Postgresql**

Huawei Technologies Company Limited

2012 Research and Development Laboratory

Distributed Database Team

Markham, Ontario, Canada

Prepared by

Mingze Xu

ID 20806115

2B Computer Science

August 23rd, 2020

**MEMORANDUM**

To: Evaluator

From: Mingze Xu

Date: August 23rd, 2020

Re: Work Report: Analysis of the Demand and Solution of

Parallel Recovery in Postgresql

As we agreed, I have prepared the enclosed report, "Analysis of the Demand and Solution of

Parallel Recovery in Postgresql" for my 2B work report and for Huawei Technologies Company Limited. This report is my second report and has not received academic credit.

The distributed Database team of Huawei Technologies is managed by Xun Xue and aims to develop the next generation of the database management system. My job as a software developer required me to implement new features, as well as stabilizing the product and improving the performance. This report aims to improve the speed of crash recovery by introducing parallel recovery in an opensource database management system, Postgresql.

The Faculty of Mathematics requests that you evaluate this report for command of the topic and technical content/analysis. Following your assessment, the report, together with your evaluation, will be submitted to the Math Undergrad Office for evaluation on campus by qualified work report markers. The combined marks determine whether the report will receive credit and whether it will be considered for an award.

Thank you for your assistance in preparing this report.

*Mingze Xu*

# Summary

This report introduced the basis of recovery in Postgresql, identified the demand for performance improvement in Postgresql recovery, and issued an approach of parallel recovery.

By analysis of the native design of Postgresql recovery and replication, the potential performance improvement could be achieved by parallelizing the recovery process. Then parallel replay in MariaDB is researched and analyzed. However, the method used in MariaDB cannot be directly applied to Postgresql due to the difference in recovery facility and logging style. After digging into the design of Postgresql recovery design, an approach of parallel recovery is issued, which dispatches all redo logs in terms of hash on the page number of the page that the redo log records operate on, in order to avoid possible conflicts in parallel replay process. This report also analyzes the disadvantage of the issued approach.

# Introduction

This section aims to provide necessary background information about database recovery and replication and the approach that Postgresql takes on replication, as well as the identification of the problem of inefficiency in the Postgresql replication process.

## Database Recovery and Replication

Database recovery often refers to the ability of DBMS (database management system) to recover data when an intentional or accidental system failure happens. It has to ensure that no data is lost due to such failures. In general, two types of backups: physical backup and logical backup are used in database recovery. Physical backups copy all the necessary files that contain database contents and store them in another physical location. Logical backups save information in the form of operations instead of physical data, such as a sequence of SQL statements. The main advantage of physical backups is the significantly higher recovery speed compared to logical backups on large databases since it only involves file copy. However, physical backups can only be performed while the database server has been stopped or locked, for the sake of data consistency. Thus physical backups are always performed on a replication of the database.

Replication enables data from one database server (primary) to be copied to another database server (standby). It can be treated as a real-time copy of the whole database. The use of replication is wide. For example, while new data is generated on the primary, analysis can be taken place on the standby without influencing the performance of the primary. A standby can be promoted to a primary when failures happen on the original primary server or it received commands from the administrator. The processes above are called "failover" and "switchover" accordingly. As mentioned before, a physical backup can only be performed when the server is offline, however, if we perform the physical backup on the replica, then the server can keep running without being influenced.

The database replication can either be physical i.e., log-shipping or logical i.e., command-shipping. The database replication can also be synchronous, wherein an applications wait time includes changes in the originator node and time to safely commit in the replica or can be asynchronous, where an application gets a response immediately after the data is safely committed in the originator node, and originator takes the responsibility of asynchronously committing the data on standby Replication can also be done on distributed database clusters, ensuring the availability of a fully functional standby cluster in event of failure of the primary cluster. (HUAWEI TECHNOLOGIES. (2017). WO 2017/122060 Al. “PARALLEL RECOVERY FOR SHARED-DISK DATABASES”)

## How Postgresql Maintains Replication

This section will introduce how Postgresql uses the transaction log to perform recovery and maintain replication.

Transaction Log or XLog is an implementation of WAL (Write-Ahead Logging). Postgresql writes all modifications such as insertion, deletion or committing as history data into the transaction log in the form of XLog Records so that when the crash happens, the database server can be restarted without data loss by rolling forward on the transaction log and perform all the operations in the xlog records.

For each operation, the necessary characteristic is extracted and assembled to be an xlog record. For instance, when an insertion happens, information such as the physical location of the data page, the insertion point on the page and the whole tuple data to be inserted will be recorded. Thus this insertion can be replayed in recovery time.

XLog records are written into the in-memory buffer first and flushed into the on-disk file right before the transaction commits or aborts. LSN (Log Sequence Number) of XLOG record represents the location where its record is written on the transaction log. LSN (Log Sequence Number) of the record is used as the unique id of XLOG record.

The streaming replication of Postgresql is implemented on the basis of transaction log shipping. A primary server continues to send Xlog data and then, each standby server replays the received data immediately. Therefore standby server could always contain the same data with the primary server with a bit lagging behind.

# Existing Problems

This section aims to identify the existing problems regarding the performance of recovery and replication in Postgresql and dig in the root cause of the problems.

## High Recovery Time Objective (RTO) Demand in application

DBMS is widely used in almost every field of business such as finance, computer science, etc. With the exploding amount of data being processed in the same time period, the stability of DBMS becomes increasingly significant. One vital measurement of stability is the RTO, which represents the expected time of a database being recovered to the original state after an accidentally or intentionally crash happens. Customers of DBMS such as banks, data analysis companies require RTO as low as possible since even a short period of database unavailability could cause a potentially huge amount of loss.

## Waste of resource in serial recovery

Postgresql supports multi-process statement processing, however, during the process of recovery, only one process is responsible for reading out the xlog and replay recorded operations. After it finished recovery the database can be considered as restarted and other run-time threads could become alive to process statements. Thus the time taker by this single node to recover the cluster is proportional to the size of xlog that is yet to be applied, which is a considerable amount of time, especially under the circumstance of high concurrency and processing speed, since the size of xlog will bloat. The same problem exists in replication as well since replication depends on xlog record reading and replaying.

# Analysis

This section aims to analyze the feasibility of parallel recovery in Postgresql. Parallel recovery in another widely used open-source DBMS MariaDB will be referred to.

## How MariaDB Handle Parallel Recovery?

Recall that Postgresql does not log SQL statements in xlog, instead, the instruction of modifying a specific row is logged. This logging method is often named "row-based logging". Replication and Recovery in MariaDB by default uses mixed logging, which is a combination of SQL-statement-based logging and row-based logging. Statement-based logging is used by default, but when the server determines a statement may not be safe (eg. CREATE TABLE, TRUNCATE TABLE) for statement-based logging, it will use row-based logging instead. (MariaDB, Mixed-logging, Retrieved from https://mariadb.com/kb/en/binary-log-formats/#mixed-logging)

In-order parallel replication are introduced in MariaDB 10.1.3 (MariaDB, , Parallel Replication, Retrieved from <https://mariadb.com/kb/en/parallel-replication/>). In in-order parallel replication mode, any transactional DML (INSERT/DELETE/UPDATE) is allowed to run in parallel. They are dispatched to different threads and replayed independently. This may cause conflicts, for example, when two statements are modifying the same row. In this case, the latter of the two statements is rolled back, so that the former could proceed. Non-transaction DML and DDL, which are mostly row-based logged, is not safe to be replayed parallelly since it cannot be rolled back. Thus those statements are not applied in parallel.

## Can the same method be applied to Postgresql?

As mentioned above, Postgresql uses row-based logging for all operations, which means only instruction of modifying a specific row is logged. During recovery, the specific row will directly be modified without being locked, which makes it difficult to identify if a given row is modified by two transactions at the same time. Thus, even transactional DML cannot be run in parallel using the same method in MariaDB.

## A possible approach to the problem

Since the native structure of xlog makes it hard to detect conflicts in parallel recovery, it is critical to avoid conflicts in advance. Notice that conflicts only happens when the same row is modified by multiple threads at the same time, then a possible approach is to dispatch all xlog records by hashing the page number of the page where the row being modified is located, as a result, all the xlog records operating on the same page are grouped together and assigned to the same process. Then each process replays the record assigned to it sequentially from the oldest to the latest. That could guarantee that no conflicts will happen between two different processes since they have to operate on different pages.

Furthermore, even non-transactional DML and DDL replay can be parallelized in this method after decoupling from the rollback feature. For example, when replaying the CREATE TABLE command, the xlog record can be dispatched by the page number of the first page of the table. However, this kind of recovery needs to be synced with other threads since all operations on this table must be replayed after the table being created.

## Disadvantages of the approach

Since the xlog records are strictly dispatched by hashing on the page number, then it could not handle some extreme cases. For example, when most of the operations are operated on the same page while only a few operations are on other pages, one process has to deal with most of the xlog records, which could significantly drop the performance of parallel replay.

# Conclusions & Recommendations

After analysis on both MariaDB and Postgresql recovery design, the conclusion can be drawn that different approach is required in Postgresql. A possible method is to dispatch all xlog records according to the page number so that conflicts could be avoided. Although this method will lose performance improvement in some extreme cases, it could save recovery time in most cases since the redo workload is parallelized.

# References

[1] HUAWEI TECHNOLOGIES. (2017). WO 2017/122060 Al. “PARALLEL RECOVERY FOR SHARED-DISK DATABASES”. Retrieved from https://patentimages.storage.googleapis.com/65/e8/0c/32b4ee42fd81cb/WO2017122060A1.pdf.

[2] MariaDB (2015), Mixed-logging, Retrieved from https://mariadb.com/kb/en/binary-log-formats/#mixed-logging

[3] MariaDB (2015), Parallel Replication, Retrieved from https://mariadb.com/kb/en/parallel-replication/

# Acknowledgments

I would like to sincerely thank my manager, Sherman Lau, for helping me to brainstorm the ideas for this report, providing advice on the structure of this report, and proofreading this report.