Replicated Database Engine in Clickhouse

Val Baturin
Faculty of Computer Science
Higher School of Economics
Moscow, Russia
val@baturin.me

***Abstract*—Nowadays it's important to have a reliable place to store data. However, even the most reliable place on Earth can fail. Clickhouse database management system is a choice of many applications to be their storage layer. And in Clickhouse there's a way to replicate data by using a replication table engine. The replication is handled just fine, although restoring the data must be done manually. The paper proposes an implementation of a database engine which will keep track of all replicas and make recovery as seamless as possible. The new database engine relies on Zookeeper as a synchronization mechanism and utilizes replicated tables functionality.**

***Index Terms*—database, Clickhouse, Zookeeper, replication**

I. Introduction

Clickhouse [1] by its nature is a distributed application which means that several instances of the same application are spawned on different nodes. The main reason to have Clickhouse up and running is to store and analyse big chunks of data. Although, it is not necessary to actually have more that one node running, the problem of data availability might only be solved using several nodes. In order to have data available in a case of node fail, in particular, disk fail, the only way to not lose data is to have a copy of it somewhere. It's a common practice to fully restore all of the data from out of service disks by using copies of the data. The process of copying data to somewhere else and use it in a case of a need is called data replication.

Data replication has its merits, however, it makes an application which supports it more complex in many ways. An example of a complication is node coordination, which will be discussed later.

From the user point of view, Clickhouse is a database management system. Users might not know to data is exactly processed and stored as long as they have an interface to interact with it using a client such as a read-eval-print loop or an API endpoint. In spite of that, users need to be sure that their data is stored somewhere reliably and in a case of emergency, such as a disk failure, the data can be restored so it will be available later and not lost.

Speaking of replication, the actual bytes that have to be replicated are incapsulated by several layers of abstractions from disk drivers to client CLI with a replicate keyword supported.

Clickhouse handles replication by using a replicated tree family of table engines. However, it's still problematic for users to maintain replicas. One of the main disadvantages of the current approach is no automatic recovery after a node fail exists. Users have to manually restore replicas which is redundant work.

This paper proposes a solution for Clickhouse replication problem by using Replicated Database Engine. Compared to table engines, database engines are a higher-level abstraction of actual data and responsible for storing metadata of tables either persistently or in-memory. Managing the same database from different nodes requires thought-out node coordination.

II. Literature Review

In the world of distributed computing node coordination was always a severe problem. The problem reduces to a consensus problem, which can be solved using the consensus protocol. The first consensus protocol which was proven and explained is Paxos [3].

However, there's an edge case for Paxos consensus protocol which in its essence is used for state machine replication which requires a workaround [5]. Unfortunately, the workaround results in poor performance.

The only suitable solution to still maintain performance is to use a different approach rather than state machine replication. One of the discovered alternatives is to use Zap protocol which is designed for primary-backup systems. Such protocol is used in Zookeeper [4]. Luckily, Zookeeper is supported by Clickhouse, and therefore it's possible to use it as a node coordination mechanism.

Zookeeper API allows one to create nodes (znodes) in a tree. A node might contain some data. Tree structure reminds a file system, with nodes being similar to files. Despite being similar to a distributed file system, Zookeeper has a different purpose. First of all, it's a service for coordinating processes of distributed applications. This design decision was taken because of several reasons. Using a file system like structure it's easier to serve different applications having just only one Zookeeper cluster available. Furthermore, the one application can use Zookeeper subpath in a tree and inside the subpath have its subpaths for different needs.

Clickhouse has its wrappers to interact with Zookeeper.

Speaking of actual engine implementation, Clickhouse source code [2] defines IDatabase abstract class which has requirements for database engines. And there's a list of existing engines in Clickhouse, each of them is inherited from IDatabase directly or stacked on different engines with the bottom being IDatabase.

III. Methods

Compared to existing engines, Replicated Database Engine would leverage a Zookeeper cluster. Zookeeper would be used as synchronized metadata storage.

Implementation of a new database engine for Clickhouse requires an understanding of the current code base related to the engines. Currently, it has several different engines for different needs of users. The new replicated database engine must obey the current requirements for database interface and bring simplification of the management of replicated tables created in a newly proposed engine instance.

Let's suppose there's a need to instantiate a replicated database using of Database Engine in Clickhouse. Since there's no point in having replication if there's only one node available, it's reasonable to assume that several nodes are available and will be involved in the process. Firstly, it must be specified that a database is replicated by choosing the replicated engine on creation. Secondly, to allow flexibility, a user might want to choose which Zookeeper path is used for a replicated instance. Thirdly, each of replicas must be distinguishable from another. It can be accomplished by using different suffixes in a Zookeeper path. The base prefix must be the same to differentiate from other replicated engine instances. Such an implementation strategy would allow creating replicas ad hoc and avoid predefined structure.

Each of the replicated tables in the engine will be created and stored on each replica. In a case of a node failure, it would possible to restore all data, because replicated table engine keeps track of them in zookeeper.

Compared to table engines, database engines keep track of all tables created inside an instance, that data inside tables. If a node of the database engine fails, it would be possible to restore all of the data it had before no matter which tables belonged to the node. Even more, the data will be up to date and synchronized through all the tables' replicas. To implement all of that, the proposed engine will be sufficient.

In order to be sure the following patch is indeed reliable and doesn't break anything, software testing is used. The current test suite of Clickhouse must not be diminished and even more, it must be extended to cover a new engine testing.

To make navigation within the code base more rapid, ctags software might be used. Ctags produces an index file which is supported by most text editors. Any text editor would be suitable for writing code for Clickhouse.

The primary target of compilation is the Linux family of operating systems. In order to avoid cross-compilation, one of the mentioned operating systems is needed to be installed and used as a development tool. GCC compiler for C++ should be used.

As one of any Clickhouse improvements, the one must be incremental and based on the latest version of the source code. Clickhouse as an open source project and it interacts with its contributors using git. Git project is hosted on GitHub, and GitHub features such as Issues and Pull Requests and used. To interact with other contributors and maintainers of the Clickhouse Project comments in issues and pull requests must be used as a communication tool.

IV. Results

Clickhouse is written in C++ language. Therefore, the new engine for Clickhouse must be written in the same language.

Final implementation of the Replicated Database Engine expected to be compilable and correct.

The changes to the source code must be in the form of a pull request. All critics and suggestions must be addressed and discussed if it is needed. The style guide must be the same as the one used in Clickhouse project.

The proposed software update must not break the current engines and other loosely-coupled components of the codebase. Furthermore, it must not break any part of the software.

The code for integration, functional, and unit tests must be written. Performance tests are optional, however, the patch must not result in decreasing performance of the current engines and Clickhouse operation in general.

The end user of the software must be able to create a replicated database instances on different nodes and with a Zookepeer path of a common prefix on each node, and using the newly proposed engine create replicated tables such that one table will be replicated on the given cluster of nodes. The cluster of nodes is defined by the end user. The cluster must be a subset of nodes running the replicated databases of a common prefix.

V. Conclusion

Proposed implementation of the engine allows eliminating currently needed efforts for users to manage replicated tables in Clickhouse, and the improve user experience.

Proposed methods allow implementing the engine so that it would fit the current codebase.

Methods provided in this paper should guarantee that adding Replicated Database Engine is a safe update in terms of backward compatibility, and doesn't break the existing qualities of Clickhouse. Nor does it reduce performance.

References

[1] *Clickhouse Documentation*. Clickhouse.tech, https://clickhouse.tech/docs/en/

[2] *ClickHouse/ClickHouse: ClickHouse is a free analytics DBMS for big data*. GitHub, https://github.com/ClickHouse/ClickHouse

[3] Leslie Lamport, *"Paxos Made Simple"*. 2001.

[4] Hunt P, Konar M, Junqueira FP, Reed B. *ZooKeeper: Wait-free Coordination for Internet-scale Systems*. In USENIX annual technical conference 2010 Jun 23 (Vol. 8, No. 9).

[5] https://cwiki.apache.org/confluence/display/ZOOKEEPER/Zab+vs.+Paxos