TIMESCALEDB CLUSTER DESIGN

HDB++ Cluster Topology with TimeScaleDb

Abstract
Designing a suitable cluster deployment topology for the HDB++ archival system based on TimescaleDb/PostgreSQL

Stuart James
sjames@esrf.fr

Table of Contents

1 Revision

Revision history.

Date	Author	Comment
01/07/19	S James	Initial Draft

2 Figures

3 Overview

The core of the HDB++ archival system is the database cluster. The cluster must be able to maintain both acceptable levels of read/write performance and high levels of availability to serve both control system data events and user queries. High availability is a common theme when deploying critical services with many existing solutions available.

This iteration on the HDB++ archival cluster is based on TimescaleDb, which is itself an extension to PostgreSQL. Therefore the design is built around the concepts, terminology and ideas within the PostgreSQL and TimescaleDb community. It is assumed the user is familiar with the terminology from both PostgreSQL and TimescaleDb software stacks.

4 Requirements

We can easily list several important requirements from the cluster. We will attempt to select software and deployment methods to satisfy each.

- Provide maximum availability. The database cluster is required to archive events at all times in an uninterrupted manor. If downtime occurs, then the cluster is required to recover as quickly as possible.
- Performant read/write query requests. The system must be both fast enough to cope with all the events generated by the Tango control system, and able to return data to users in q reasonable time.
- Scalable deployment. The design should not make future expansion difficult, where possible expansion should just be a case of adding an additional cluster node.

5 HDB++ Cluster Design

The cluster design is split into three sections:

- Topology Design. A short review of the design pattern used.
- Software Architecture. Selection of software to meet the needs of the cluster architecture.
- Deployment. A brief note on deployment and any weaknesses to be addressed.

5.1 Topology Design Model

The following diagram illustrates the topology of a high availability cluster. We have split the model into five areas.

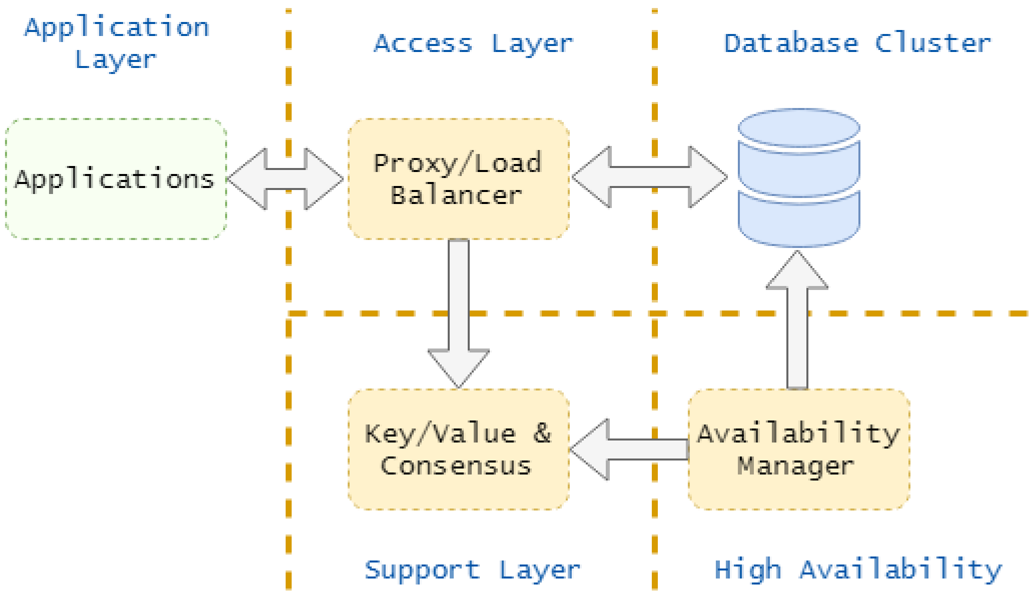

Figure 1 Topology

Unlike the more generic pattern, an additional component has been added, the Support Layer. This is in recognition of the software required to help support the other layers functionality.

The following sections give a brief overview of each area.

5.1.1 Application Layer

Representing application and user interactions with the database cluster. Traditionally these applications would be given the hostname and port number of the actual database. This is no longer possible when running a cluster with the possibility that the contact points may change due failures, maintenance, or expansion.

Instead they are now given the hostname and port of the proxy in the access layer. This will direct them to the correct nodes within the cluster.

5.1.2 Access Layer

Abstracts the contact point into a well-known hostname and port (similar to a virtual IP). Behind this abstraction the actual database cluster can be configured and run on any server without affecting the users in the Application Layer.

It is also possible to have some element of load balancing present in the Access Layer, for example, read/write request splitting, directing certain queries to certain nodes etc.

The software in this layer helps address both Performance and Accessibility.

5.1.3 Database Cluster

The actual deployed database servers as a cluster. While it is possible to support many different configurations, here we concentrate on a master/replica strategy. The following nodes will exist in the database cluster:

- Master – This node is the current lead. It is set up to stream its data into the replica nodes.
- Replica – One or more additional databases that the master will replicate data to. Each replica is ready to replace the master when required.

Each time the master fails, a replica gets promoted to the position of master. Therefore the number of consecutive failures without recovery the cluster can tolerate is equal to the number of replica nodes.

We can utilise this master/replica pattern to the advantage of the cluster performance:

- The master will handle all write requests. Event data will be inserted via the master.
- The replica nodes will serve read only users of the database cluster. Any application wishing to query data will be directed to these replica nodes.

This database cluster model helps address both the Availability and Performance requirements. We can scale the cluster by adding more replica nodes.

5.1.4 Support Layer

Any additional services to support software in the other layers.

5.1.5 High Availability

High availability represents the role of software which is managing the nodes in the database cluster. Its role is to ensure the database nodes are restarted, or switched when failures occur. Its function will help address the Availability requirements.

5.2 Software Stack Architecture

TimescaleDb has a helpful blog post where they have completed an evaluation of various solutions, and recommend several pieces of software. Since this design builds on their database, we build our cluster mainly around their evaluation and recommendations.

5.2.1 Software Selection

Following is some brief comments on the software.

5.2.1.1 Patroni

A Python tool for building high availability clusters. It manages PostgreSQL configuration, can be configured to handled replication, backups, failovers and restores etc. Deployed alongside the database node it is managing.

5.2.1.2 Etcd

A fault tolerant, distributed key-value store. Via Patroni this will store the state of the database cluster here and check periodically for changes. Usually deployed as a cluster to ensure availability.

5.2.1.3 HAProxy

A single endpoint for the Application Layer to connect to. HAProxy forwards a connection to whichever node Patroni is reporting as the master. We can utilise HAProxy to also forward read only requests to the replica nodes.

Load balancing the request across the cluster will help performance.

5.2.2 Software Architecture

Combining the topology design with the chosen software we can construct the software architecture as follows:

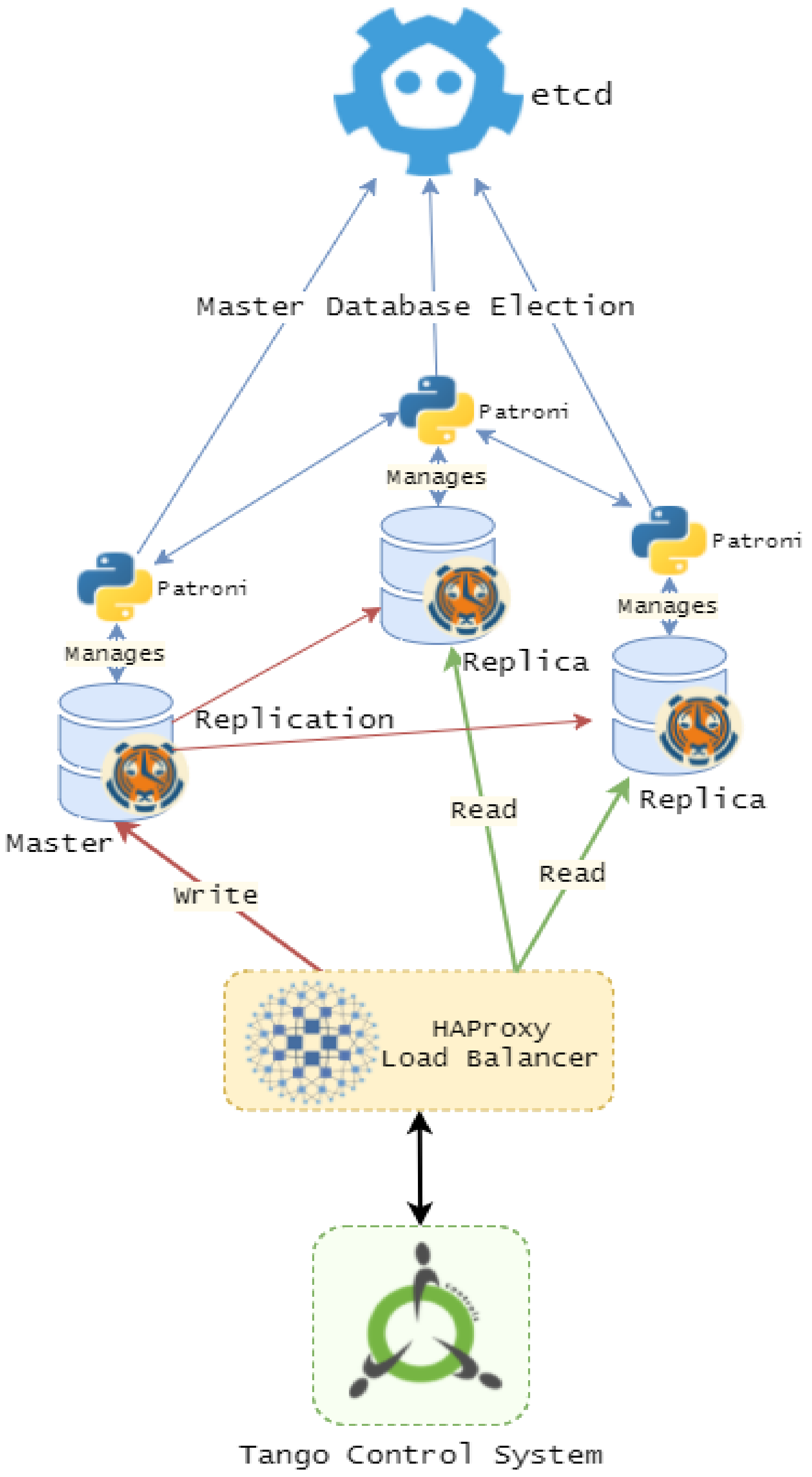

Figure 2 Software Architecture

Some comments on the diagram:

- HAProxy is splitting the read and write requests to different parts of the cluster. This is done by offering two connection points, i.e. different ports on the host, which map to either the master node, or replica nodes. This strategy should be reinforced by configuring the database with both read/write and read only user accounts for the connecting applications.
- Each TimescaleDb node has a Patroni deployment alongside it to manage it.
- The master is elected via Patroni setting a key in etcd.

- HAProxy, etcd and Patroni all represent possible failure points.

5.2.3 Support Software

To facilitate the dynamic reconfiguration of the proxy when the cluster elects a new master node, we need some additional software. On election of a new master, the proxy must update its rule on how to distribute the connection requests.

We can use confd to watch for changes in etcd, and perform this update:

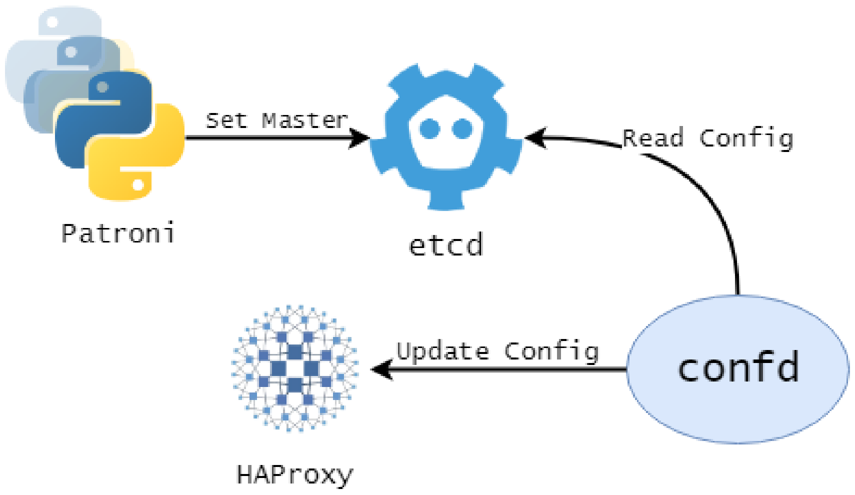

Figure 3 confd integration

Here confd watches for cluster configuration changes in etcd, and updates the HAProxy configuration when it detects a change. This means read/write and read only connections are now forward to the correct database nodes within the cluster.

5.3 Deployment Model

The two obvious models of deployment are container or sever based.

5.3.1 Virtualised/Real Servers

This deployment can be logically generalised as following:

- Deploy Tango Device Servers relating to the HDB++ install (Event Subscriber etc) on their own server or servers. This removes them as a possible influence on the stability of the cluster.
- Each TimescaleDb/Patroni deployment must be deployed on its own server. As per database deployment advice, try to minimise other running software on the server. This both increases stability and allows for a node to be down without consequence to the rest of the system.
- HAProxy, etcd, confd can be viewed as HDB++ service software, they can be deployed together or across several servers.

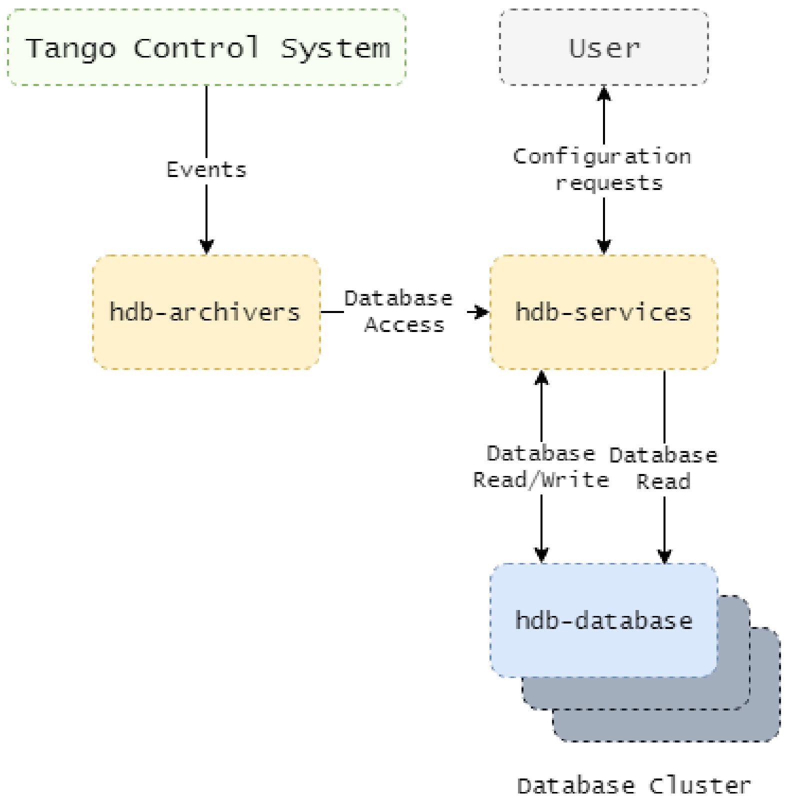

Figure 4 Server Deployment

Some comments:

- hdb-archivers aggregates the Tango Device Servers into one or more servers. These connect to HAProxy deployed on hdb-services.
- hdb-services either aggregates all the service software onto a single server, or represents a cluster of servers.
    - A single server is simple to manage and maintain, but is a failure point. Can be partially mitigated by setting the software to auto restart via, for example, supervisord or systemd.
    - Multiple servers will increase fault tolerance but also the complexity and maintain aspects.
    - Virtual machines mitigate the cost aspect, and allow resources to be allocated to the server when required.
    - There is no reason why it should not be possible to initially deploy all service software together on a single server, and migrate some to individual servers at a later date.
- hdb-database is a combined TimescaleDb and Patroni install per node.

As noted previously, the software supporting the cluster itself can present an additional failure point. We can help address this as follows:

- Run under systemd (Debian/Ubuntu), or distribution appropriate system process (recommended, this is simple and quick to setup).
- Run using supervisord.
- For etcd we can run multiple etcd nodes as a cluster.

5.3.2 Docker/Kubernetes

Support for this deployment model is available via a Docker image combining both Postgresql (with TimescaleDb extension) and Patroni, called Spilo. Other elements of the architecture are also available Dockerized, or can be Dockerized very quickly for custom requirements.

These containers would allow ether individual container deployments, or better a Kubernetes deployment.

This deployment method is not explored further here.

5.3.3 Scaling

It both deployment cases, scaling the database deployment is limited for write performance, but more flexible for read performance.

- Scaling write performance is only possible by improving the hardware. The current open source/community release of TimescaleDb does support splitting the TimescaleDb hyper-tables across multiple servers. This feature is only planned for the enterprise edition.
- Read performance can be scaled rapidly by adding more replica nodes to handle the queries.

5.4 Additional Considerations

When deploying the system, the following items not detailed here must also be considered:

- Backup. A system to archive data in case of emergencies, where a full rebuild of the database cluster is required.
- Monitoring tools. Activities such as checking performance, searching for bottlenecks or reporting cluster health can be supported by a number of freely available tools.

6 Conclusion

The initial deployment can be quite complex in terms of skill and resources. A simple deployment first, expanding at a later date as familiarity and experience with the software increases would recommended. This could involve:

- Aggregating services onto a single deployment server
- Deploying a single etcd instance, or a cluster on a single server
- Deploying all HDB++ Tango Device Servers on a single server
- Keeping the number of database nodes low until the system has been confidently tested.