IraHive HA Framework

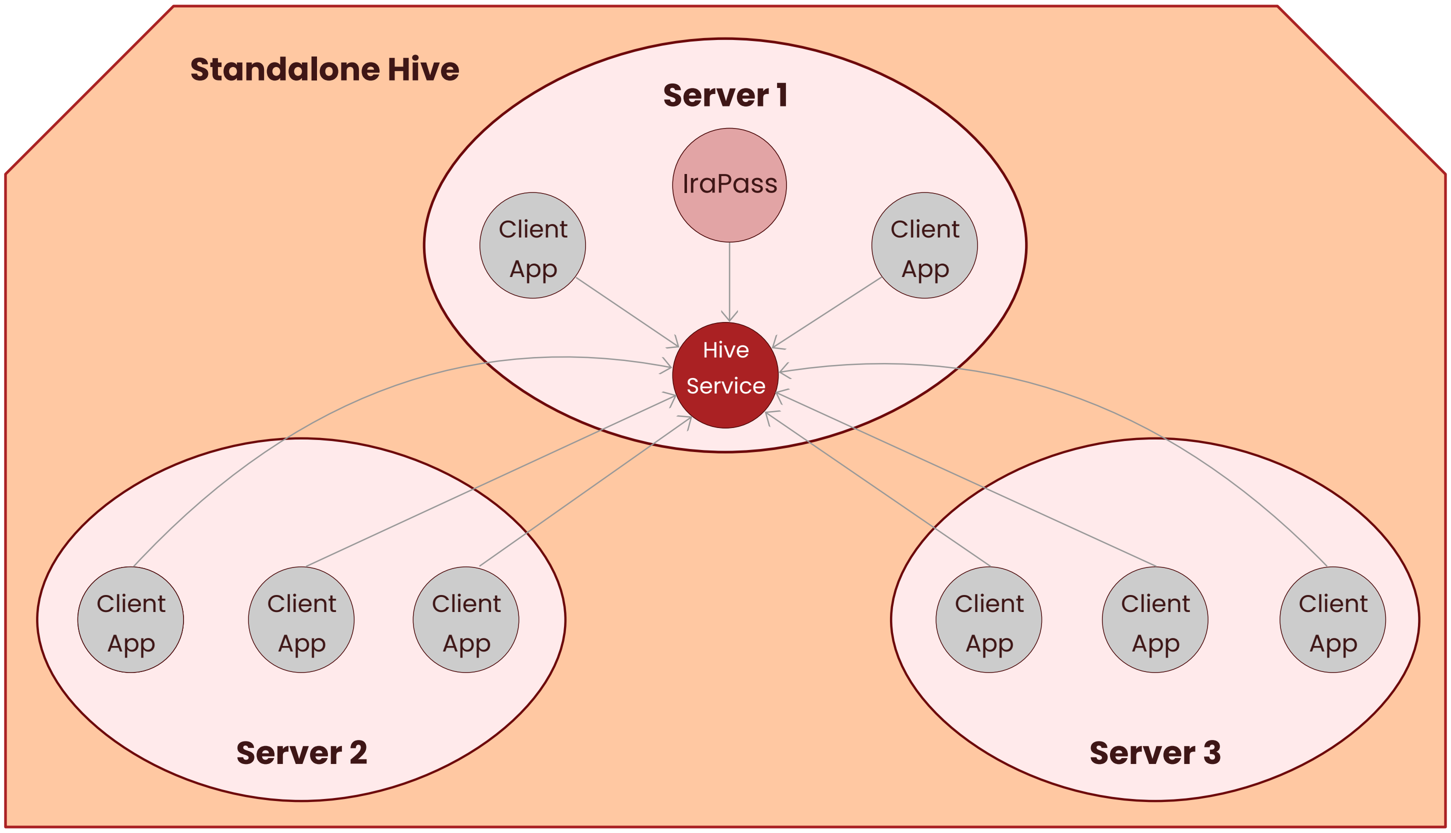
IraHive is an interprocess messaging framework with redundancy and load balancing features. Architects and Programmers always wish for a seamless way to communicate between programs, without resorting to complicated APIs.

The wish list also contains automatic service discovery, message priority, RSA encryption, high availability, selective buffering if the destination is not ready, or sent by simple name or role, and sent to one destination or many or all. IraHive was designed to cater to all those needs using very simple and lightweight APIs. IraHive is simply referred to as Hive most of the time.

The term Hive refers to how a bee hive operates.

The IraHive framework also contains a transient real-time embedded database named Orb. The data stored in Orb resides within the application's heap memory, so it eliminates even network latency while making SQL queries. The database tables are automatically synced across applications subscribed to the tables and the built-in trigger support means that each application can set trigger functions for insert/ update/delete on interested tables. It is called a transient database, because it is primarily used for fast changing data which needs to be kept synchronized across multiple applications. Think of it as an universal global variable or distributed global variable, which can be managed as a SQL table, with indexing and trigger support.

1 Standalone Hive



A hive should have one or more Hive service instances. If the hive has only one Hive service instance, then all the hive clients will be connected to that single Hive service in a star configuration. This basic hive setup won't have any redundancy if Hive service is down.

When a Hive service starts, it starts advertising its presence on a known multi-cast address or via redis DB accessible to all clients. The client applications that wish to join a common hive will listen on the same multi-cast address or read from redis DB for Hive services that are running. Once a Hive service makes its presence known, IraPass (floating license manager) will be the first client to join the hive and issues core licenses to the hive. After the licenses are issued, the rest of the clients applications can connect to Hive service and become part of the hive. It usually takes 5-10 seconds for all the client applications to join the hive. It is possible to have multiple hives in the same machine or subnet using different multi-cast addresses or hive names.

After joining the hive, the applications can send messages to each other just using a name as the destination address. The name could be the id or the role of the application that was used while joining the hive. The message can be sent to following category of destinations:

BY_ADDRESS: A Hive client creates a client address when it joins the hive, generally it will be <ServiceName>_<MachineName>, if the service can only have one instance per machine. If the client can have multiple instances per machine, it will also contain process ID. To send a message by address, the sender must know this address. This is generally used for stateful transactions, while replying asynchronously to a request sender.

BY_ID_ONE: If a request can be serviced/handled by any instance of an application, this is the preferred method. The Hive service will load balance all the instances of the destination application and send the message to one of the instances. If a reply is required, it always uses by_address method to reply back to the requester.

BY_ID_ALL: This method is generally used for informational purposes or to synchronize/replicate data across multiple instances of an application.

BY_ROLE_ONE: A given application can have multiple roles. If a request can be serviced/handled by any instance of applications that share a common role, this is the preferred method. The Hive service will load balance all the instances of the destination applications with that role and send the message to one of the instances. If a reply is required, it always uses by_address method to reply back to the requester.

BY_ROLE_ALL: This method is generally used for informational purposes or to synchronize/replicate data across multiple instances of applications with a given role.

BROADCAST: This method is used if a message needs to be sent to all the clients that are in hive. This is very rarely used.

1.1 Control Messages

Every client receives certain control messages which they can use for various purposes.

- **Roster:** This is the first message received by the client upon joining the hive. This message contains the presence information of every client that is already in the hive. The presence info contains the id, role, address and start-time of the client.
- **Join:** This message is received by every hive member when a new client joins the hive. The message contains the presence info of the new hive member.
- **Leave:** This message is received by every hive member when a client leaves the hive.

1.2 Performance Considerations

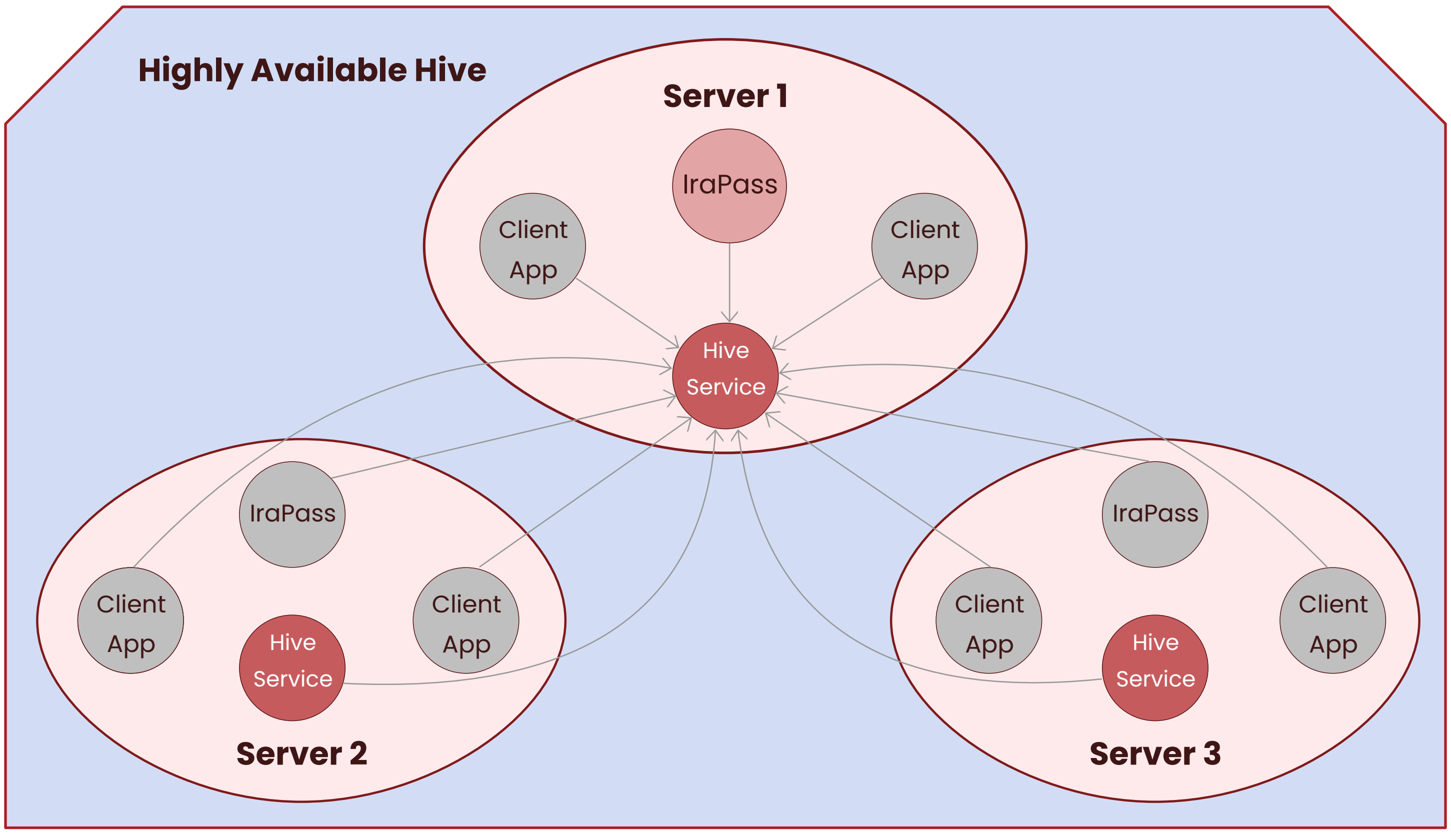
Hive was designed to be extremely fast with minimum overhead. Some of the salient features are as follows:

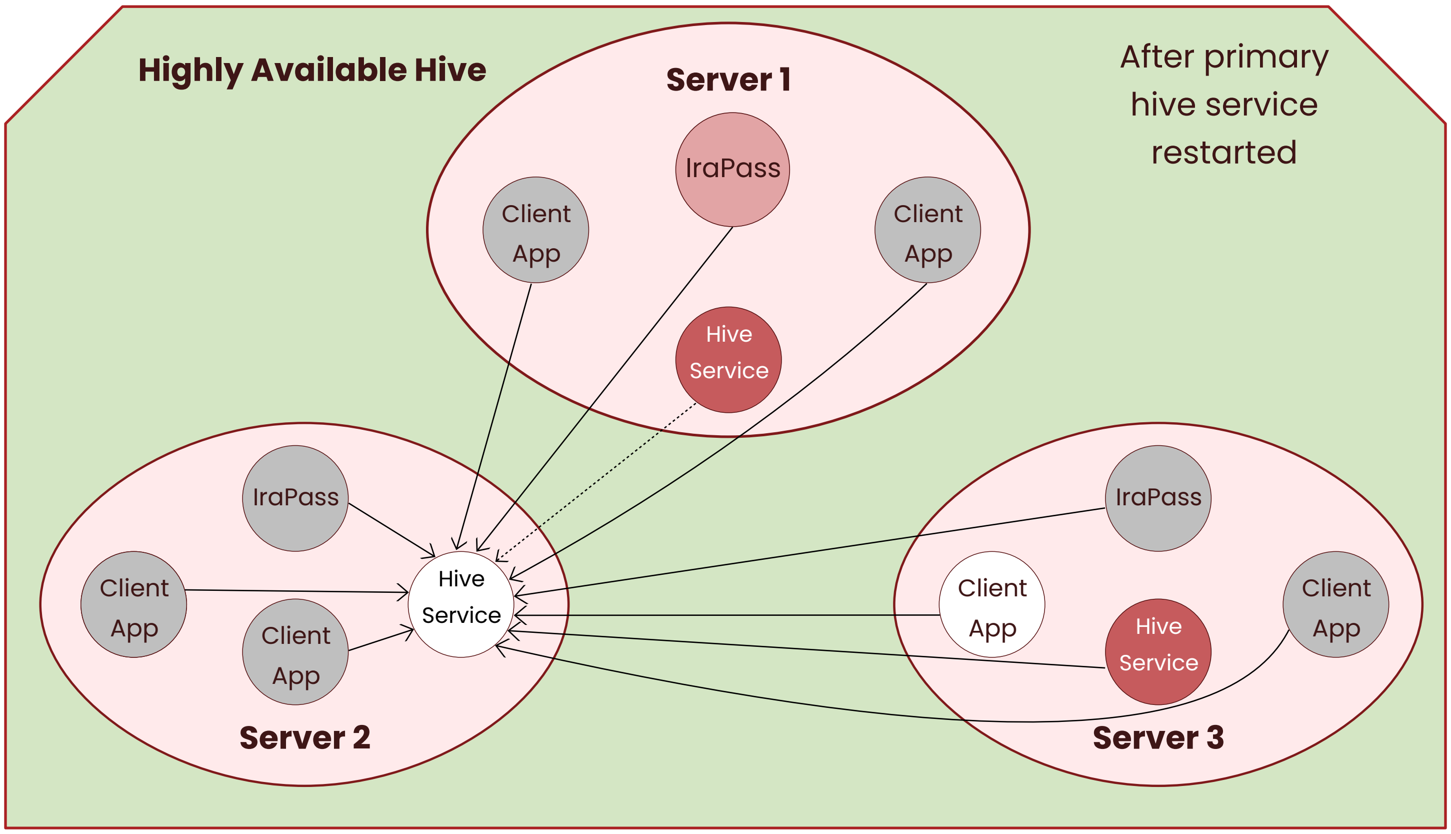
- All the socket connections between Hive service and the clients are continuously kept open. So there is no delay in opening or closing sockets for sending each message.
- Hive messaging uses every CPU core available via multi threading to maximize performance.
- Critical messages like control messages have the highest priority. Applications can set priority to override the default one.
- Messages can be selectively marked for guaranteed delivery.

2 Highly Available (HA) Hive

Hive was designed to be extremely fast with minimum overhead. Some of the salient features are as follows:

- All the socket connections between Hive service and the clients are continuously kept open. So there is no delay in opening or closing sockets for sending each message.
- Hive messaging uses every CPU core available via multi threading to maximize performance.
- Critical messages like control messages have the highest priority. Applications can set priority to override the default one.
- Messages can be selectively marked for guaranteed delivery.





When a Hive instance is started, it also looks for older instances of Hive service. If an older instance is found, it will connect to the oldest Hive service instance (primary) as a client, and stays around as a backup. If the oldest Hive in the hive goes down, all the clients will connect to the next oldest Hive service and re-create the hive. The images in the previous page show the transition that occurs in a highly available hive. When the original primary Hive service comes back, it will connect to the new primary Hive instance as a backup instance.

Sometimes a machine can lose network connectivity, leading to a split hive. Until the network connection is restored, there will be a split hive, which allows the applications to retain connectivity at least within the disconnected machine. When the network connection is restored, the two parts of the split hives join together to become a single hive like before.

3 Transient embedded real-time distributed database

Orb is an embedded serverless real-time distributed database with redundancy load balancing features. Architects and Programmers always wish for a seamless way to share transient data (quick changing) across machines and programs, without compromising on performance or simplicity. This becomes very important when transient state information of various types must be highly synchronized across multiple instances of the same or different applications. This cannot be used for persistent storage.

For example, in an enterprise telephony system, the state of an agent at any instant should be available to the Softphone, Dialer, Dashboard, CallQueue, etc. This is an example of different applications needing perfectly synced access to the same transient data. Similarly, all transient data of multiple instances of CallQueue require perfect synchronization to achieve redundancy and load balancing.

Traditionally, this kind of synchronization is achieved by storing the transient data in a centralized location in an in-memory database. This would usually create a single point of failure, and require far more complex design to achieve redundancy and load balancing. Not to forget, the network latency and the delay caused by mutually exclusive locking of the data would make pull-type synchronization quite in-efficient. **Here the applications that modify transient data are responsible for updating the central location. And the applications that need the latest copy of transient data must pull it from the central location.**

To solve this problem, a different approach was considered using the Hive Framework. An application can create a named database table within the process, and add itself and other applications as subscribers. This is essentially located in the heap memory of the process. However, once the table is created, the Hive messaging architecture will replicate this table creation asynchronously to every instance of all the applications in the hive that are subscribed to that table. When the table is modified (insert/update/delete), the changes will also be replicated asynchronously to every instance of all the applications in the hive that are subscribed to that table. These replications are handled by the Orb layer, and are invisible to the process. **Here the applications that modify transient data are responsible for updating only local data within the process. The applications that need the latest copy of transient data will always find it within the process.**

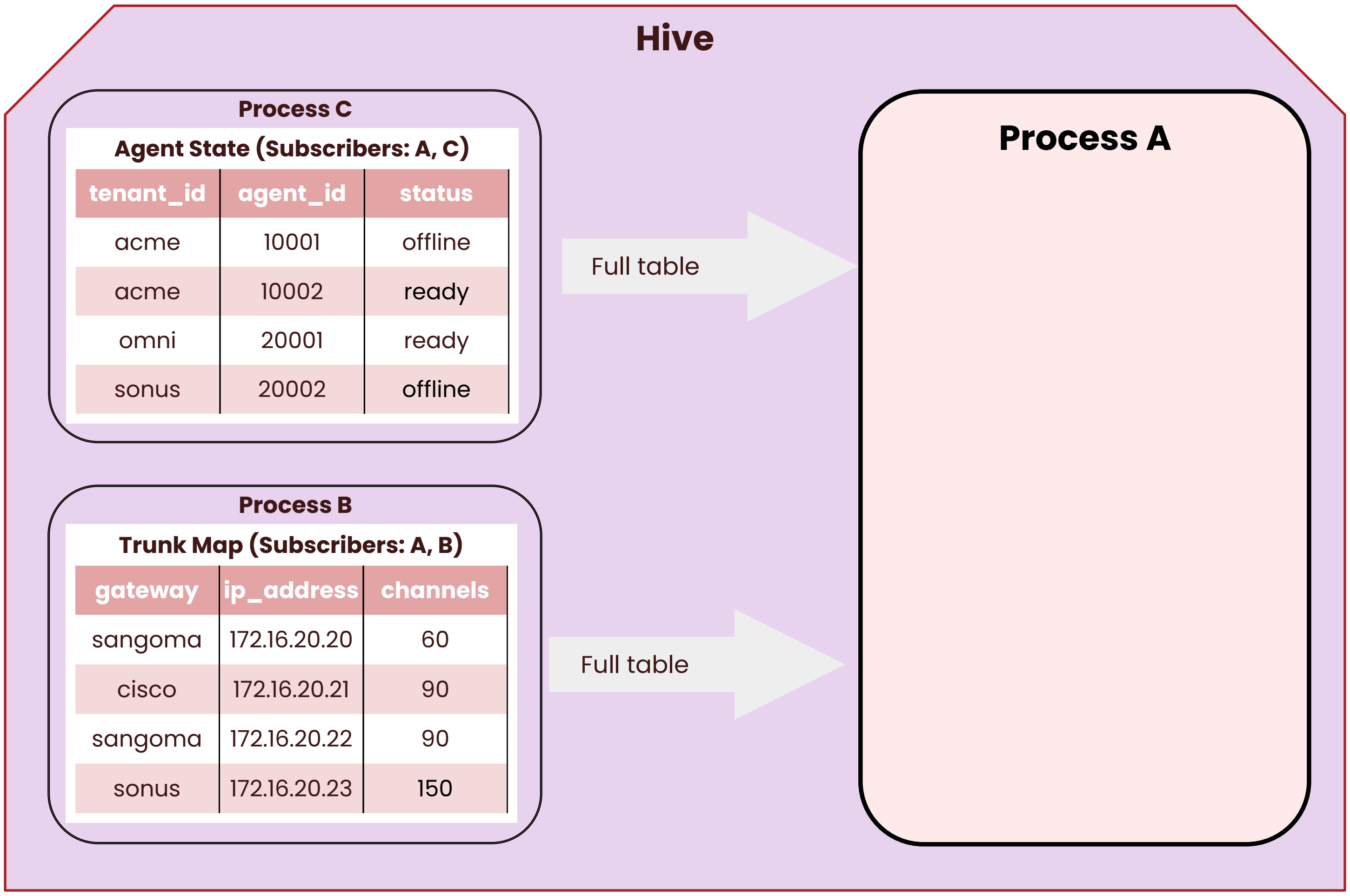
Contrast this with the scenario in the previous paragraph. Since the processes always work with the copy they already have within, the performance improvement is phenomenal.

3.1 Orb Replication Concept

This section will detail some of the use cases of replication, to give a better idea about the concept. Whenever a new table is created within a process, replication instruction is sent to all the subscribers of the table, so that same table is created at every subscriber process.

3.1.1 What happens when a process instance is started?

Let's assume one instance of process B and C are already running, with one table each that they have already created.



Agent State (Subscribers: A, C)

tenant_id	agent_id	status
acme	10001	offline
acme	10002	ready
omni	20001	ready
sonus	20002	offline

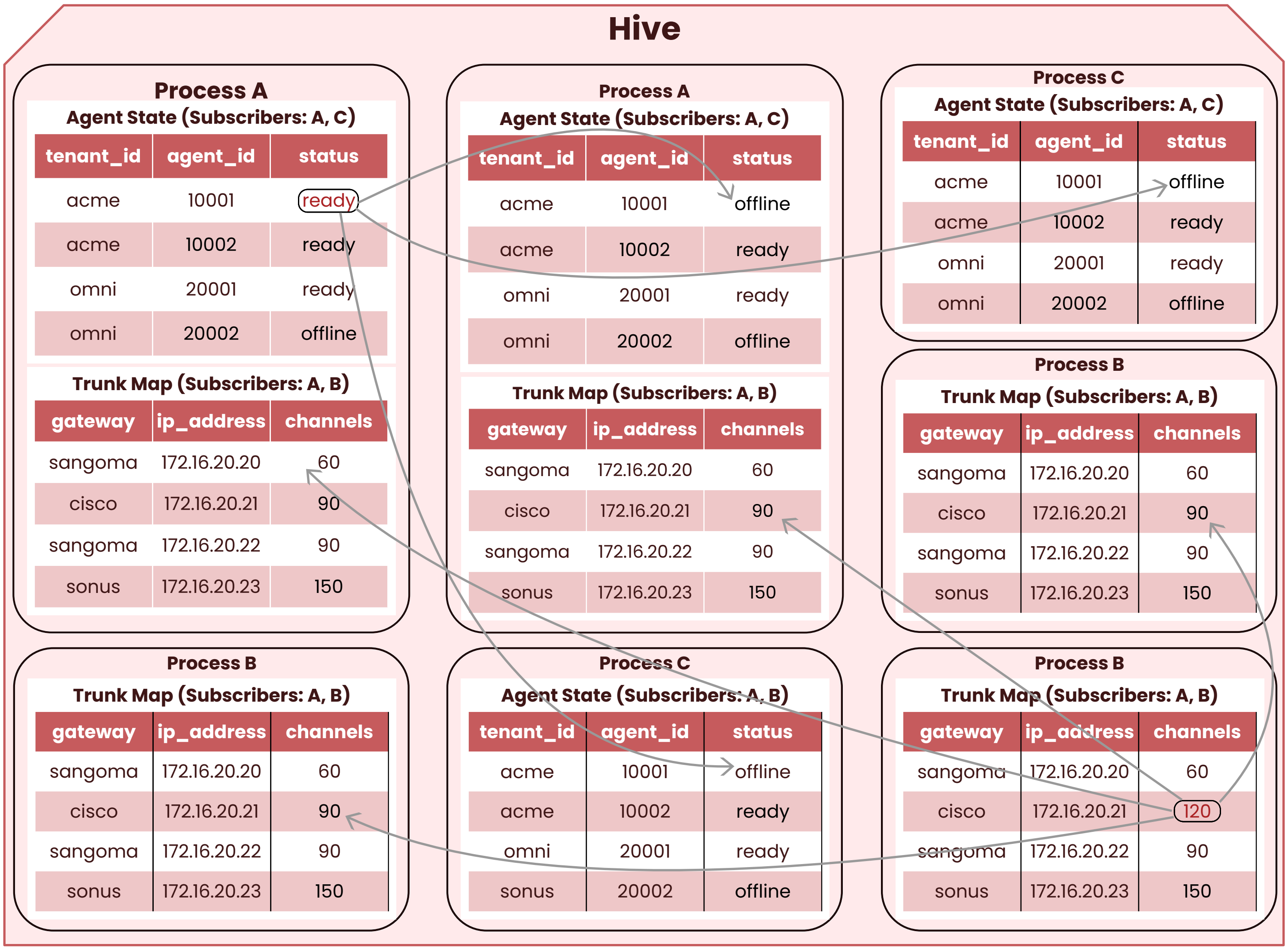
Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	90
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

Now a new instance of A is started. Process A is a subscriber of both the tables. When process A joins the hive, both process B and C receive a join message with the address of process A. Since A is a subscriber of a table in B, the oldest instance of process B will send the entire copy of table to process A. Similarly, since A is a subscriber of a table in C, the oldest instance of process C will send the entire copy of table to process A. The subscription list maintained by the creator of the table. Same logic will apply to any number of subscriptions that the newly launched instance will have. If the hive already has a process that created the table, the oldest instance of the process will send the full copy of the tables to the newly arrived subscriber process.

3.1.2 How changes to transient data gets replicated?

The below image shows an example of 3 applications sharing 2 tables between them.



Process A

Agent State (Subscribers: A, C)

tenant_id	agent_id	status
acme	10001	ready
acme	10002	ready
omni	20001	ready
omni	20002	offline

Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	90
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

Process A

Agent State (Subscribers: A, C)

tenant_id	agent_id	status
acme	10001	offline
acme	10002	ready
omni	20001	ready
omni	20002	offline

Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	90
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

Process C

Agent State (Subscribers: A, C)

tenant_id	agent_id	status
acme	10001	offline
acme	10002	ready
omni	20001	ready
omni	20002	offline

Process B

Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	90
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

Process B

Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	90
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

Process C

Agent State (Subscribers: A, B)

tenant_id	agent_id	status
acme	10001	offline
acme	10002	ready
omni	20001	ready
sonus	20002	offline

Process B

Trunk Map (Subscribers: A, B)

gateway	ip_address	channels
sangoma	172.16.20.20	60
cisco	172.16.20.21	120
sangoma	172.16.20.22	90
sonus	172.16.20.23	150

- Table “Agent State” is subscribed by process A and process C. Has 4 copies in the hive.
- Table “Trunk Map” is subscribed by process A and process B. Has 5 copies in the hive.
- There are 2 instances of Process A in the hive. Each has a copy of both tables.
- There are 3 instances of Process B in the hive. Each has a copy of “Trunk Map” table.
- There are 2 instances of Process C in the hive. Each has a copy of “Agent State” table.

When a change is made to a row in “Agent State” table in Process A, the Orb layer will send a single replication message to the Hive service, targeted at all the instances of A and C processes. The Hive service will find all the qualifying instances and send the messages. This results in 3 other copies of the table getting the update. Even insert and delete actions get replicated in a similar fashion. In the next example, a change is made to a row in “Trunk Map” table in Process B. The Orb layer will send a single replication message to the Hive service, targeted at all the instances of A and B processes. This results in 4 other copies of the table getting the update.

3.1.3 Standard features of Orb

- **SQL query support:** Programmers can use simple SQL where clauses to select the rows from the tables.
- **Optional Triggers:** The programmers can assign triggers on insert/update/delete/publish on any table. If the change is made to a table in process A, which gets replicated on the same table in process B, it can trigger a callback in process B. This is a very useful feature in event driven programming models. For example, a web based dashboard can update the screen only when a trigger is received.
- **Primary key support:** The tables can assign primary keys using single or multiple keys. Can also be used for indexed search.
- **Indexing support:** Programmers can define any number of indexes using one or more keys. Very useful while searching large tables. There are even diagnostic tools to tune the indexing performance.

- **Redundancy:** If the hive contains multiple instances of the same process with a bunch of tables, they will have copies of identical data. If one of the instances crashes, the data is intact in other running instances. If the crashed instance comes up again, it will receive the full set of all the latest tables from the oldest instance of that process. This often lets the architects design applications that will never dip into disk based databases after initialization.
- **Load Balancing:** Since every instance of a process has identical data, the requests can be sent to any instance of a process using BY_ID_ONE or BY_ROLE_ONE category destination.
- **High performance:** Just like the Hive layer, the Orb uses every CPU core available via multi-threading to maximize performance.
- **Virtual tables:** The programs can publish into virtual tables, which are not stored, but only replicated to subscribers of those virtual tables. It works similar to multi-casting. The subscribers get the published data, as long as the process has a trigger on publish event for that virtual table.
- **Portability:** Supports Windows & Linux on physical servers, cloud servers, docker containers and even tiny devices like raspberry pi. Either Redis or multi-cast support is required for service discovery.

4. Language Support and Licensing

Hive natively supports python, C# and C++. Licenses are counted by the number of CPU cores in each server running the Hive service. Both node-locked and cloud licensing is supported. However, node-locked is only recommended for physical servers and VMs where the instance doesn't change after every restart. Environments using DHCP or autoscaling or containers must stick with cloud licensing.