Highly Available IceStorm

IceStorm offers a highly available (HA) mode that employs master-slave replication with automatic failover in case the master fails.

On this page:

- IceStorm Replication Algorithm
- IceStorm Replica States
- Client Considerations for IceStorm Replication
- Subscriber Considerations for IceStorm Replication
- Publisher Considerations for IceStorm Replication

IceStorm Replication Algorithm

HA IceStorm uses the Garcia-Molina "Invitation Election Algorithm" [1] in which each replica has a priority and belongs to a replica group. The replica with the highest priority in the group becomes the coordinator, and the remaining replicas are slaves of the coordinator.

All replicas are statically configured with information about all other replicas, including their priority. The group combining works as follows:

- When recovering from an error, or during startup, replicas form a single self-coordinated group.
- Coordinators periodically attempt to combine their groups with other groups in order to form larger groups.

At regular intervals, slave replicas contact their coordinator to ensure that the coordinator is still the master of the slave's group. If a failure occurs, the replica considers itself in error and performs error recovery as described above.

Replication commences once a group contains a majority of replicas. A majority is necessary to avoid the possibility of network partitioning, in which two groups of replicas form that cannot communicate and whose database contents diverge. With respect to IceStorm, a consequence of requiring a majority is that a minimum of three replicas are necessary.

An exception to the majority rule is made during full system startup (i.e., when no replica is currently running). In this situation, replication can only commence with the participation of every replica in the group. This requirement guarantees that the databases of all replicas are synchronized, and avoids the risk that the database of an offline replica might contain more recent information.

Once a majority group has been formed, all database states are compared. The most recent database state (as determined by comparing a time stamp recorded upon each database change) is transferred to all replicas and replication commences. IceStorm is now available for use.

IceStorm Replica States

IceStorm replicas can have one of four states:

- Inactive:
  The node is inactive and awaiting an election.
- Election:
  The node is electing a coordinator.
- Reorganization:
  The replica group is reorganizing.
- Normal:
  The replica group is active and replicating.

For debugging purposes, you can obtain the state of the replicas using the `replica` command of the `icestormadmin` utility, as shown below:

```
$ icestormadmin --Ice.Config=config
>>> replica
replica count: 3
1: id:         1
1: coord:      3
1: group name: 3:191131CC-703A-41D6-8B80-D19F0D5F0410
1: state:      normal
1: group:
1: max:        3
2: id:         2
2: coord:      3
2: group name: 3:191131CC-703A-41D6-8B80-D19F0D5F0410
2: state:      normal
2: group:
2: max:        3
3: id:         3
3: coord:      3
3: group name: 3:191131CC-703A-41D6-8B80-D19F0D5F0410
3: state:      normal
3: group:      1,2
3: max:        3
```

Each line begins with the identifier of the replica. The command displays the following information:

- `id`
  The identifier of the replica.
- `coord`
  The identifier of the group's coordinator.
- `group name`
  The name of the group to which this replica belongs.
- `state`
  The replica's current state.
- `group`
  The identifiers of the other replicas in the group. Note that only the coordinator knows, or cares about, this information.
- `max`
  The maximum number of replicas seen by this replica. This value is used during startup to determine whether full participation is necessary. If the value is less than the total number of replicas, full participation is required.

Client Considerations for IceStorm Replication

As previously noted, an individual IceStorm replica can be in one of several states. However, IceStorm clients have a different perspective in which the replication group as a whole is in one of the states shown below:

- Down
  All requests to IceStorm fail.
- Inactive
  All requests to IceStorm block until the replica is either down (in which case the request fails), or becomes Active.
- Active
  Requests are processed.

It is also possible, but highly unlikely, for a request to result in an `Ice::UnknownException`. This can happen, for example, if a replica loses the majority and thus progresses to the inactive state during request processing. In this case, the result of the request is indeterminate (the request may or may not have succeeded) and therefore the IceStorm client can draw no conclusion. The client should retry the request and be prepared for the request to fail. Consider this example:

C++

```
TopicPrx topic = ...;
Ice::ObjectPrx sub = ...;
IceStorm::QoS qos;
topic->subscribeAndGetPublisher(qos, sub);
```

The call to `subscribeAndGetPublisher` may fail in very rare cases with an `UnknownException`, indicating that the subscription may or may not have succeeded. Here is the proper way to deal with the possibility of an `UnknownException`:

C++

```
TopicPrx topic = ...;
Ice::ObjectPrx sub = ...;
IceStorm::QoS qos;
while(true) {
    try {
        topic->subscriberAndGetPublisher(qos, sub);
    } catch(const Ice::UnknownException&) {
        continue;
    } catch(const IceStorm::AlreadySubscribed&) {
        // Expected.
    }
    break;
}
```

Subscriber Considerations for IceStorm Replication

Subscribers can receive events from any replica. The subscriber will stop receiving events under two circumstances:

- The subscriber is unsubscribed by calling `Topic::unsubscribe`.
- The subscriber is removed as a result of a failure to deliver events. Subscribers can optionally configure a quality of service parameter that causes IceStorm to make additional delivery attempts.

Publisher Considerations for IceStorm Replication

A publisher for HA IceStorm typically receives a proxy containing multiple endpoints. With this proxy, the publisher normally binds to a single replica and continues using that replica unless there is a failure, or until active connection management (ACM) closes the connection.

As with non-HA IceStorm, event delivery ordering can be guaranteed if the subscriber and publisher are suitably configured and the publisher continues to use the same replica when publishing events.

Ordering guarantees are lost as soon as a publisher changes to a different replica. Furthermore, a publisher may receive no notification that a change has occurred, which is possible under two circumstances:

- ACM has closed the connection.
- Publishing to a replica fails and the Ice invocation can be retried, in which case the Ice run time in the publisher automatically and transparently attempts to send the request to another replica. The publisher receives an exception if the invocation cannot be retried.

A publisher has two ways of ensuring that it is notified about a change in replicas:

- The simplest method is to use the `Topic::getNonReplicatedPublisher` operation. The proxy returned by this operation points directly at the current replica and no transparent failover to a different can occur.
- If you never want transparent failover to occur during publishing, you can configure your publisher proxy so that it contains only one endpoint. In this configuration, the `Topic::getPublisher` operation behaves exactly like `getNonReplicatedPublisher`.

Of the two strategies, using `getNonReplicatedPublisher` is preferable for two reasons:

- It does not involve changes to IceStorm's configuration.
- It is still possible to obtain a replicated publisher proxy by calling `getPublisher`, whereas if you had used the second strategy you would have eliminated that possibility.

The second strategy may be necessary in certain circumstances, such as when an existing IceStorm application is deployed and cannot be changed.

Regardless of the strategy you choose, a publisher can recover from the failure of a replica by requesting another proxy from the replicated topic using `getPublisher` or `getNonReplicatedPublisher`.

See Also

- IceStorm Administration
- IceStorm Quality of Service
- Active Connection Management
- IceStorm Delivery Modes
- Automatic Retries
- Configuring IceStorm

References

1. Garcia-Molina, H. 1982. Elections in a Distributed Computing System. *IEEE Transactions on Computers* 31 (1): 48-59.