HIGH AVAILABILITY AND FAILOVER TUNING IN TERRACOTTA

From the Terracotta Team

Please view the slide notes for more explanation of the slide contents.

WHAT IS FAILOVER?

- In the event of failure of an active server, a new active is elected by the surviving mirror (passive) servers
- Clients switch from the failed active server to the newly elected active server to continue performing operations

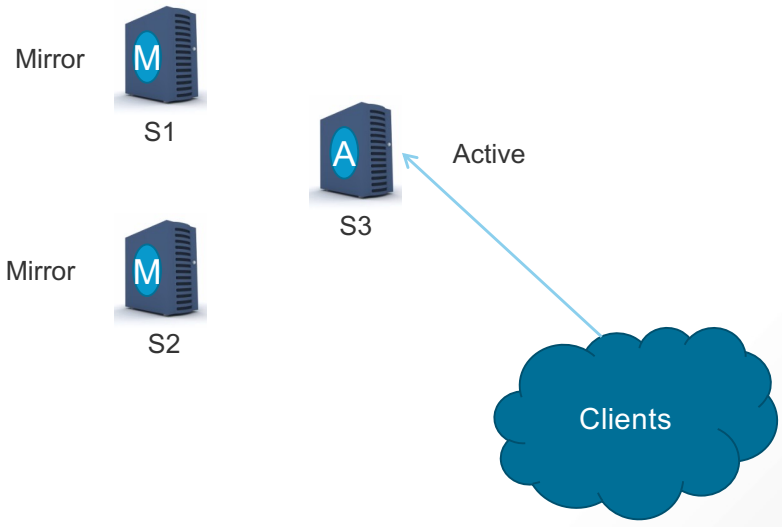
Terracotta Server Arrays are composed of one or more "stripes". For a cluster of "N" stripes, each stripe contains (approximately) 1/Nth of the data stored in the cluster.

Stripes are composed of one or more Terracotta Server processes. Within the stripe one server is the "active" server, and the other servers (if any) are mirror, or "passive", servers. Mirror servers receive data replication messages from the active server in order to keep their view of the data consistent with the active. (Mirrors/passives also constantly receive other cluster state information from the active, such that the cluster state is also consistent).

When the active server fails (crashes, or is out of contact due to a network failure, or etc.):
* the surviving passive/mirror servers begin work to elect a new active server
* clients attempt to locate an active server in order to switch to it to continue their work

This is called "fail-over".

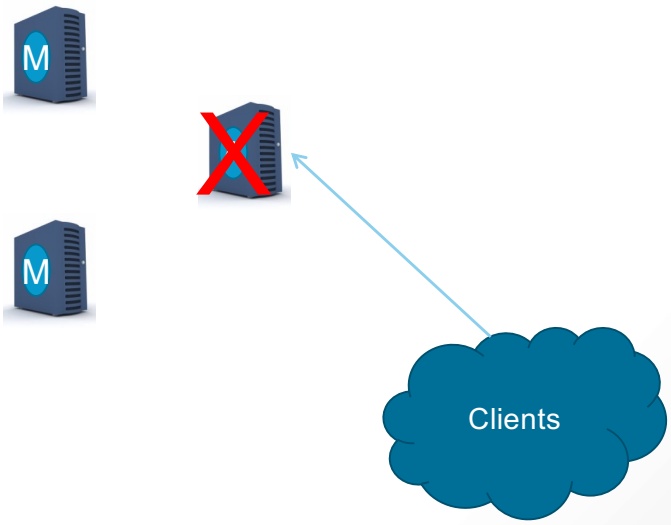
FAILOVER EXAMPLE – INITIAL STATE NORMAL OPERATIONS



This picture depicts:

* a stripe with 3 servers: S1, S2, and S3
* S3 is currently the active server, as denoted by "A"
* S1 and S2 are currently mirror servers, as denoted by "M"
* Some number of clients are connected to the active server (and presumably retrieving/storing data)

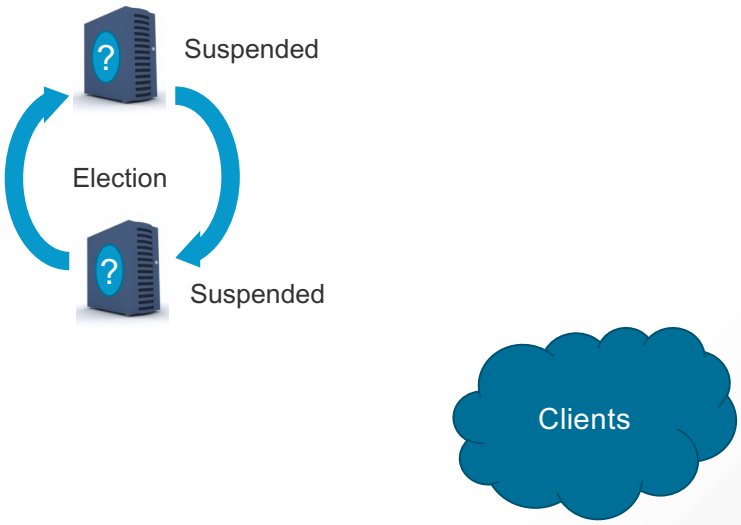
FAILOVER EXAMPLE – ACTIVE SERVER FAILS



This picture depicts:

* the active server has failed

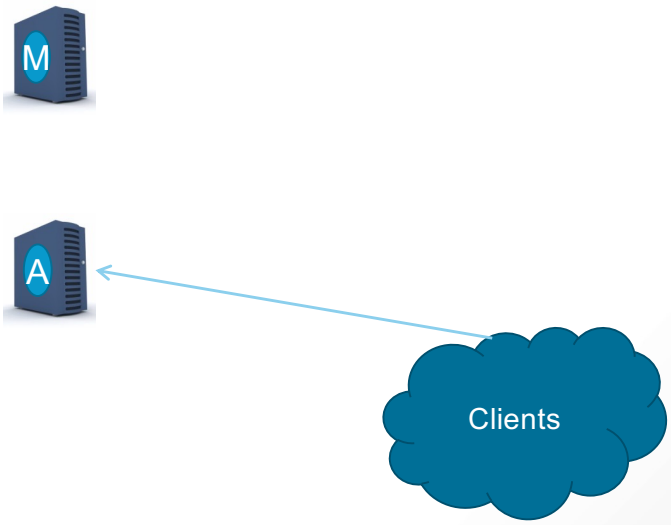
FAILOVER EXAMPLE – ELECTION IS HELD BY SURVIVORS



This picture depicts:

* The surviving servers of the stripe detect the failure of the active server and automatically suspend their activities
* The surviving servers begin an election to determine which of them will become the new active server

FAILOVER EXAMPLE – NEW ACTIVE IS ELECTED



This picture depicts:

* A new active server has been elected
* The other server has returned to the role of being a mirror/passive
* The clients have automatically connected to the newly elected active server, and resumed work

This fail-over process successfully preserved availability of the data – meaning applications using the data remain available for normal usage.
However! This eagerness to elect a new active can also lead to tricky situations, such as so-called "split-brain"

WHAT IS A SPLIT BRAIN ?

- A "Split Brain" is the situation where two or more servers (of the same stripe) are in "active" mode at the same time.
- It is a possible side-effect of the cluster's eagerness to be highly available to clients
- Caused when network partitions split a cluster into smaller subgroups (subgroups of servers that can communicate with each other, but not with the servers in the other subgroups) and each subgroup elects a new active server within them.
- A Split-brain situation is VERY dangerous to data consistency

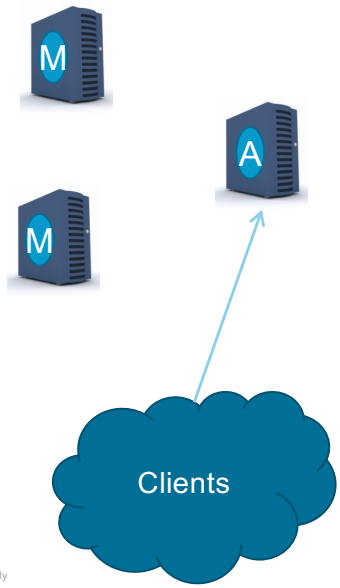
A "Split Brain" is the situation where two or more servers (of the same stripe) are in "active" mode at the same time. It is a (possible) side-effect of the cluster's eagerness to be highly available to clients.

A split-brain is typically caused when network failures (aka "a partitioning of the network") split a cluster into smaller subgroups (subgroups of servers that can communicate with each other, but not with the servers in the other subgroups). Because they can't communicate with each other, each sub-group assumes that all the other servers have failed. In order to remain available, each subgroup elects a new active server within them.

A Split-brain situation is very dangerous to data consistency, because clients that are able to connect to them can modify data, yet those changes are not replicated to the other active.

When the split-brain is healed (I.e. when the network between the multiple active servers is fixed), only one of the actives will survive a new election, and the data changes that occurred in the other active will be lost!

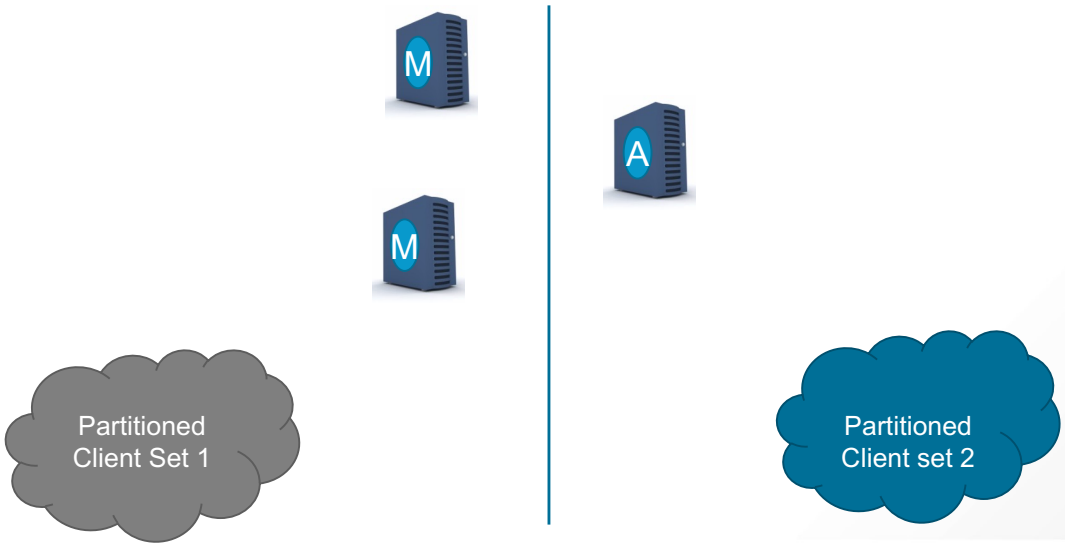
SPLIT BRAIN EXAMPLE – NORMAL OPERATIONS



This picture depicts:

* A stripe with 3 servers, operating normally, with one active server, two mirror servers, and a number of clients using the active server

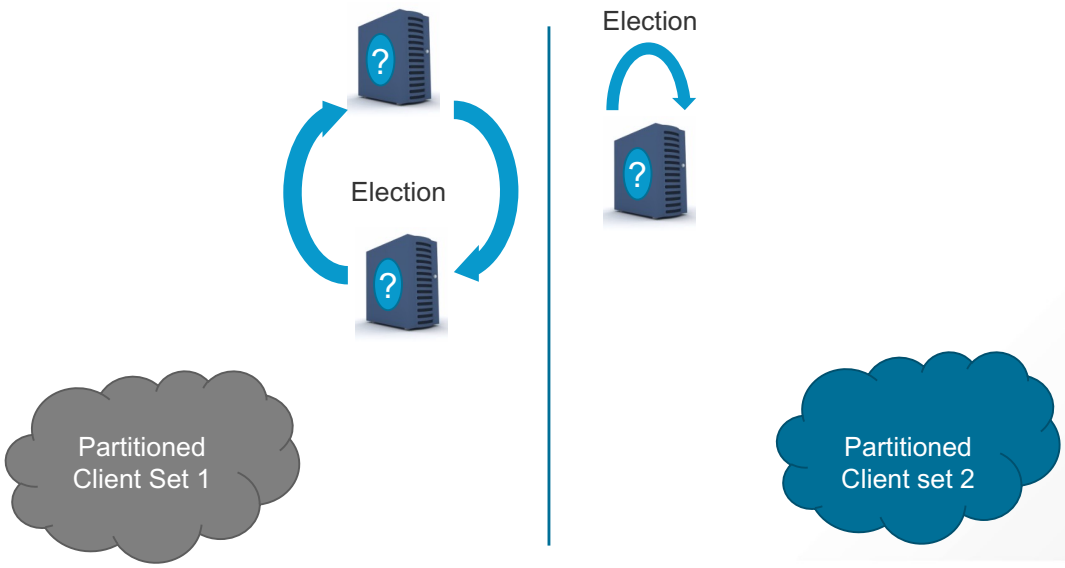
SPLIT BRAIN EXAMPLE - NETWORK PARTITION OCCURS



This picture depicts:

* A network partition (depicted by the vertical line) has occurred
* Some clients ("Partitioned Client set 2") can still "see" (communicate with) the active server
* Other clients ("Partitioned Client set 1") cannot see the active server, but they can see the mirror servers
* The mirror servers can see each other but not the active server
* The active server can see no other servers

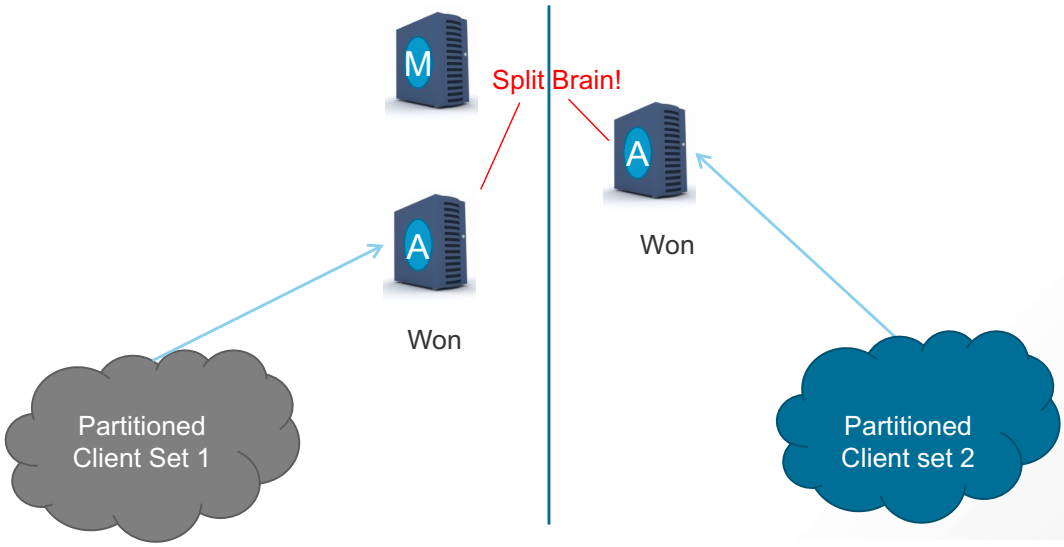
SPLIT BRAIN EXAMPLE - ELECTIONS BEGIN



This picture depicts:

* All three servers detect loss of communication with each other, and go into suspended mode (prevent client operations)
* Each subgroup of servers holds an election

SPLIT BRAIN EXAMPLE - ELECTION COMPLETES



This picture depicts:

* Each subgroup of servers elects an active server
* The clients in Partitioned Client Set 1 connect to the active server that they can reach, and resume work
* The clients in Partitioned Client Set 2 connect to the active server that they can reach, and resume work

A Split-Brain has formed!

Data consistency between the two actives will be lost if any data adds/updates/deletes are performed by the clients.

When the split-brain is healed (I.e. when the network between the multiple active servers is fixed), only one of the actives will survive a new election, and the data changes that occurred in the other active will be lost!

SPLIT BRAIN – IS IT ALWAYS A BAD THING?

Is Split-Brain always bad?

Most people would agree yes ... but it depends upon your use case!

For instance:

- Some use cases are read-only, in which case there is no risk of loss of data consistency, and the continued availability of data during split-brain is a good thing!
- Cache data is already always drifting away from consistency with the SoR … in some cases, more loss of consistency is no big deal
- etc.

AVOIDING SPLIT BRAIN, AVOIDING LOSS OF AVAILABILITY
THE CAP THEOREM

A Terracotta Server Array (TSA), being a distributed system, is subject to the constraints of the *CAP Theorem*.

The CAP Theorem states that it is impossible for a distributed system to simultaneously provide guarantees for Consistency, Availability, and Partition tolerance. (see https://en.wikipedia.org/wiki/CAP_theorem)

A TSA always seeks to be tolerant of network partitions so a choice must be made between data consistency and service availability.

FAILOVER PRIORITY CONFIGURATION

You must configure your Terracotta Server Array to favor either "availability" or "consistency" of data when situations arise that could lead to split-brain.

- To favor availability:

  ```
  failover-priority=availability
  ```

- To favor consistency:

  ```
  failover-priority=consistency
  ```

 |

software AG

The "failover-priority" cluster property can be used to "tune" which behavior you would like to favor/guarantee in the case of a potential split-brain scenario.

FAILOVER PRIORITY - AVAILABILITY

- Cluster remains highly available in the case of server failures or network partitions
- Risk of getting into a split-brain scenario exists

Favoring the "availability" of the data means that servers are much less likely to remain in a "suspended" state (more likely to promote themselves to be "active") when they lose contact with other servers.

This means that they can be used by clients to perform operations on the data. However it also means that another server may also be in active state at the same time!

FAILOVER PRIORITY - CONSISTENCY

- Mandates majority quorum for an active election
- Ensures that there will always be only one active server in a stripe irrespective of the kind of failure in the cluster – no split brain
- Sacrifices availability in certain situations

Favoring the "consistency" of the data means that servers may stay in the "suspended" state when they lose contact with other servers.

This means that they cannot be used by clients to perform operations on the data. However it also means that there is (almost) no risk of other servers being in active state at the same time.

Make sure the stripe has an odd number of voting participants so that there is no ambiguity in elections!

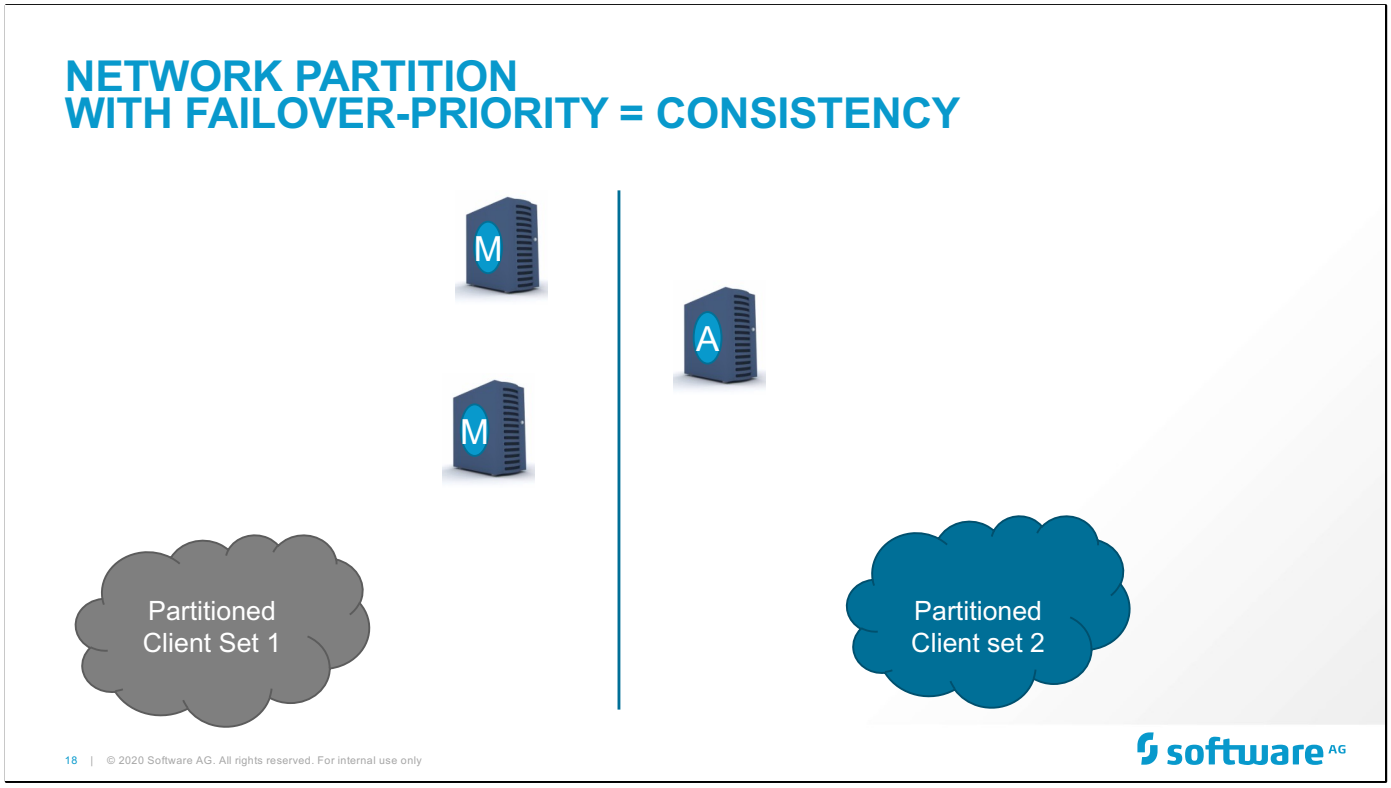
QUORUM WITH GUARANTEED CONSISTENCY

- An election is won if and only if a node gets > 50% votes
- Votes required out of total number of stripe members:
  - 5 node -> 3
  - 4 node -> 3
  - 3 node - > 2
  - 2 node -> 2

When configured for "availability", an election is won when receiving majority of votes out of all stripe nodes that can be communicated with (rather than out of the total number of stripe members).

software AG

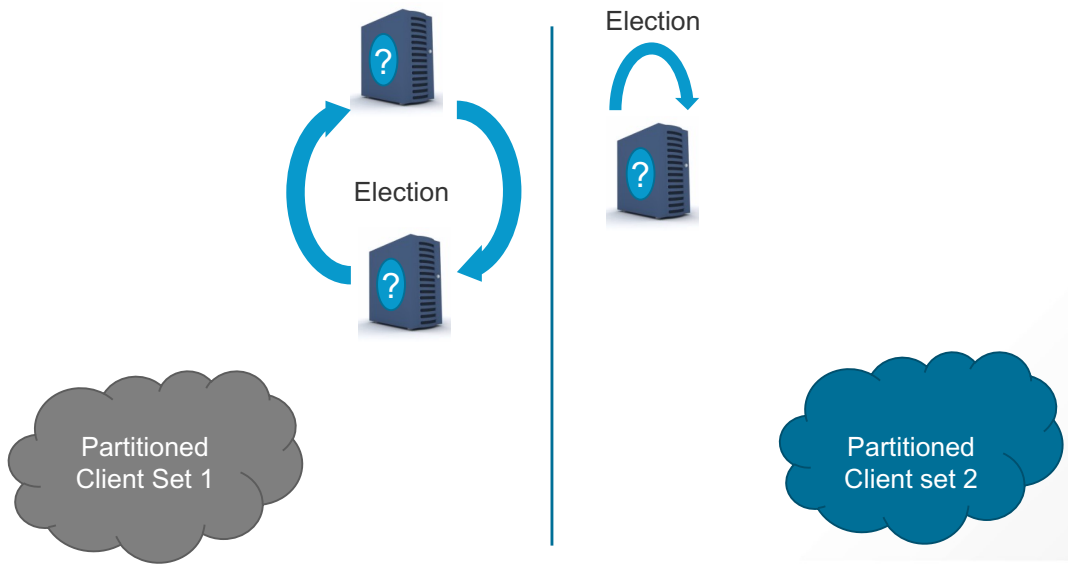
Configurations with odd numbers of TC servers per stripe (or odd number of servers + voters) is recommended to reduce the likelihood of tie votes (and servers "stuck" in "suspended" mode). More on "voters" in upcoming slides.



This picture depicts:

* A three-server stripe, with an active and two mirrors
* A network partition (depicted by the vertical line) has occurred
* Some clients ("Partitioned Client set 2") can still "see" (communicate with) the active server
* Other clients ("Partitioned Client set 1") cannot see the active server, but they can see the mirror servers
* The mirror servers can see each other but not the active server
* The active server can see no other servers

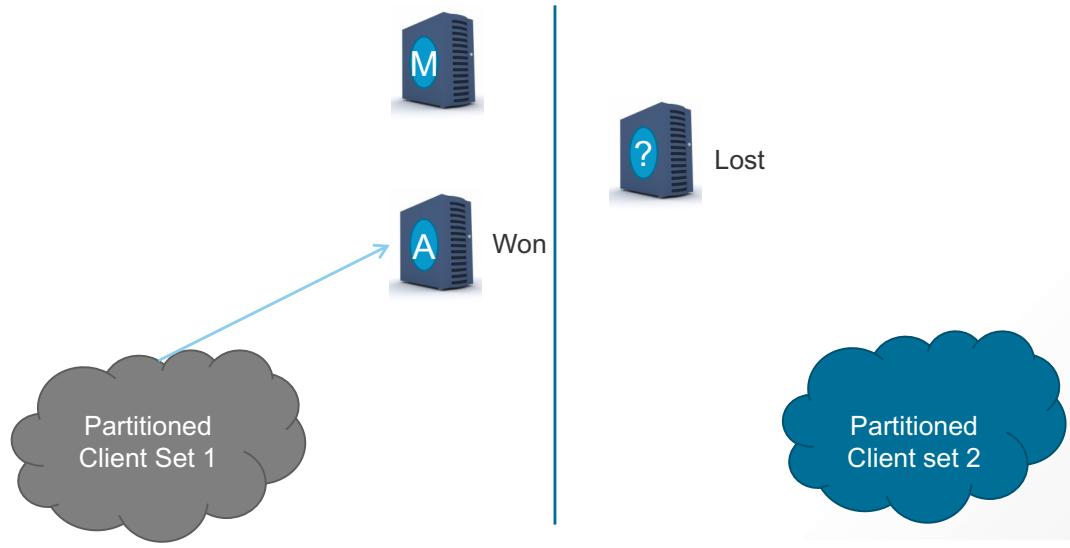
QUORUM ELECTION



This picture depicts:

* All three servers detect loss of communication with each other, and go into suspended mode (prevent client operations)
* Each subgroup of servers holds an election

QUORUM ELECTION WITH FAILOVER-PRIORITY=CONSISTENCY



This picture depicts:

* The subgroup with two servers is able to elect a new active (because it receives 2/3 votes)
* The subgroup with one server is unable to elect a new active (because it receives 1/3 votes)
* The clients in Partitioned Client Set 1 connect to the newly elected active server that they can reach, and resume work
* The clients in Partitioned Client Set 2 are unable to perform work because the only server they can reach is suspended

A Split-Brain has been avoided!
(at the cost of lost availability to some clients)

Note that the set of clients in "Partitioned Client set 2" may be zero. Note that the same may be true about "Partitioned Client set 1" !

LIMITED AVAILABILITY OF CONSISTENT CLUSTER

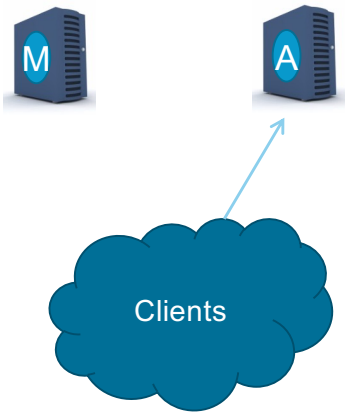
When using failover-priority=consistency, there are cases where loss of availability is possible.

Most common scenarios leading to availability loss:

- Failure of the active server in a two-server stripe
- A network partition splitting a stripe into two equal-sized subgroups

In both cases the election will result in a 50% vote, failing the >50% requirement.

2-NODE QUORUM – NORMAL OPERATIONS



This picture depicts:

* A stripe with 2 servers, operating normally, with one active server, one mirror server, and a number of clients using the active server

2-NODE QUORUM

This picture depicts:

* The active server fails, and clients lose their connection
* The surviving server runs an election

Note that the surviving server has no way of telling the difference between the network being partitioned and the other server actually not being running. Either way, all it knows is that it cannot communicate with it.

2-NODE QUORUM WITH FAILOVER-PRIORITY=CONSISTENCY

- Lost

This picture depicts:

* The surviving server receives only 50% of the vote (less than the required >50%)
* The surviving server is "stuck" in suspended mode
* The clients are unable to connect to and use any server

Because the remaining server is told not to risk creating a split-brain (via failover-priority=consistency), availability of service is lost until the failed server comes back online.

Again, note that the surviving server has no way of telling the difference between the network being partitioned and the other server actually not being running. Either way, all it knows is that it cannot communicate with it.

SOLUTIONS FOR HA FOR 2-NODE CLUSTER

- Add a third server to the stripe
  - A fine solution – adds additional resilience in case another server fails before restoration of the first failure
  - But, costs an additional machine
- Terracotta allows you to add additional, light-weight voter into the election process in order to achieve a safe quorum.
  - Voter options
    - Terracotta clients
      - Can be configured to act as "voters"
    - Stand-alone process
      - One or more light-weight "voter" processes can be ran
- Manual operator intervention can promote a suspended server to be active
  - Delayed restoration of service

software AG

Two-server stripes are prone to problems – split-brain when in "availability" mode, loss of availability when in "consistency" mode.

Two-server stripes are a very commonly desired deployment model – they add redundancy, but only incur the cost of one more machine (per stripe).

The Terracotta platform includes the concept of additional "voters". Voters are not TC Servers, but they participate in the election process, in order for there to be an additional vote in order to break ties / reach quorum.

Terracotta clients can be configured to participate as voters. In which case they run an additional service within the client application which monitors for the need to cast a vote.

A light-weight stand-alone "voter" process can be ran, which monitors for the need to cast a vote.

Multiple voters can be used so that there is high-availability of voters themselves!

FAILOVER PRIORITY – CONSISTENCY - VOTERS

Configuring the servers in a cluster to expect extra voters is simple.

The cluster is configured to expect N additional votes when elections are held (default is 0). You can run a larger number of voters than the configured number of voter "slots", in which case the "extra" voters wait for registered voters to fail, and then register themselves to take their place (resulting in HA for voters themselves).

Simply add to the "consistency" setting the number of external votes that elections will expect:

```
failover-priority=consistency:1
```


STANDALONE VOTER

Script to start a standalone voter process

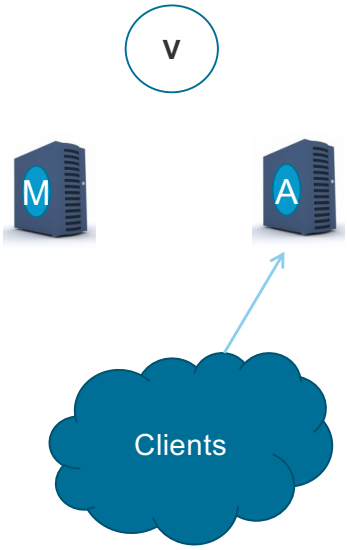
```
<install dir>/tools/voter/bin/start-voter.sh -connect-to <host:port>,<host:port>,...
```

Multi-clusters can be supported by one voter process by adding multiple "connect-to" parameters.

The stand-alone voter process executable can be found under the installation directory in the "tools/voter/bin" folder.

When running the voter process, give it the host:port for multiple servers of the cluster, to ensure that the voter is able to connect even if one is down.

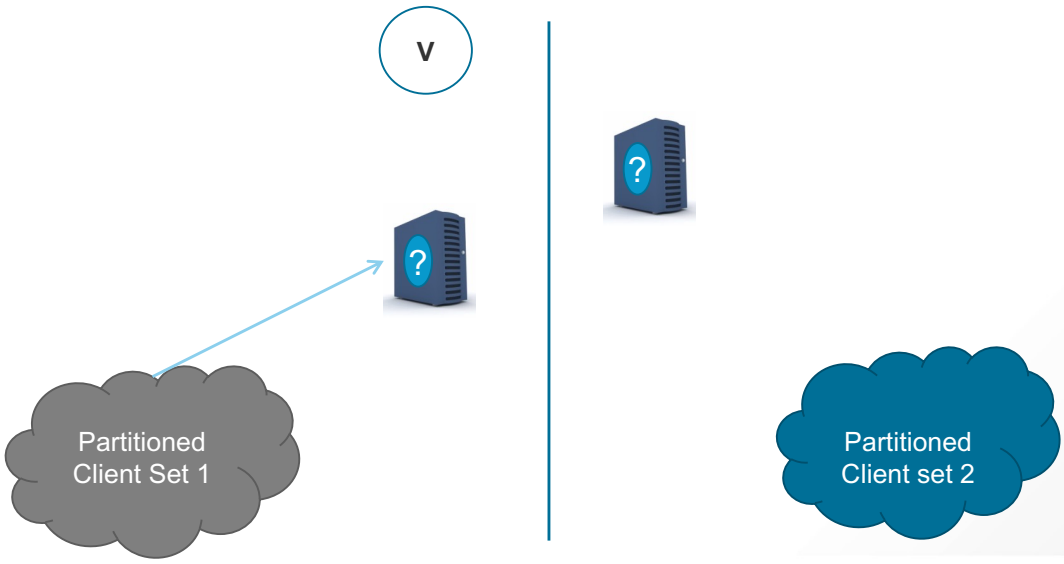
EXAMPLE WITH A STANDALONE VOTER



This picture depicts:

* A stripe with 2 servers, operating normally, with one active server, one mirror server, and a number of clients using the active server
* A stand-alone voter process monitoring the cluster

QUORUM ELECTION…

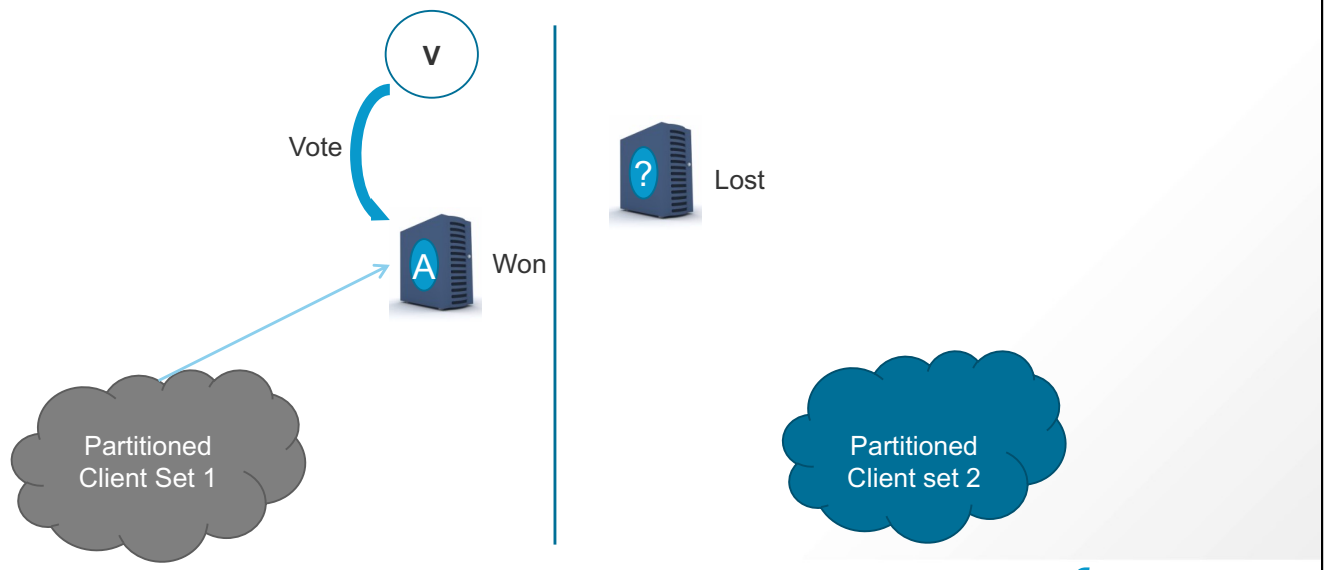


software AG

This picture depicts:

* A network partition (depicted by the vertical line) has occurred
* Some clients ("Partitioned Client set 2") can still "see" (communicate with) the previously active server
* Other clients ("Partitioned Client set 1") cannot see the active server, but they can see the mirror server
* The mirror server can no long see the previously active server
* The active server can see no other servers
* The voter can see the mirror server, but not the active server

QUORUM ELECTION…



This picture depicts:

* Both servers run an election
* The previously active server can only receive its own vote, so it loses the election and remains suspended
* Some clients ("Partitioned Client set 2") can still "see" (communicate with) the previously active server, but can't use it because it is suspended
* The voter casts a vote for the previously mirror server, so it receives 2/3 vote, which is >50%, so it wins the election and becomes active
* The clients in "Partitioned Client set 1") see the new active, connect to it, and resume work

Note that the set of clients in "Partitioned Client set 2" may be zero. Note that the same may be true about "Partitioned Client set 1" !

VOTER EMBEDDED WITHIN CLIENTS

- Server configuration:

```
failover-priority=consistency:3
```

- Client configuration:

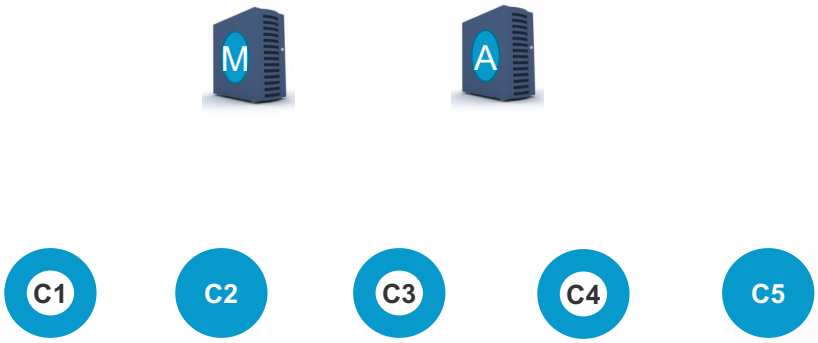
```
TCVoter voter = new TCVoter();
voter.register("my-cluster-0”, "terracotta://<host>:<port>,<host>:<port>");
```


software AG

You can specify more than 1 additional voter if you wish. Make sure that the total number of voters + number of members per stripe is an odd number!

Some sample code for instantiating a voter service within a client application.

VOTER EMBEDDED CLIENTS

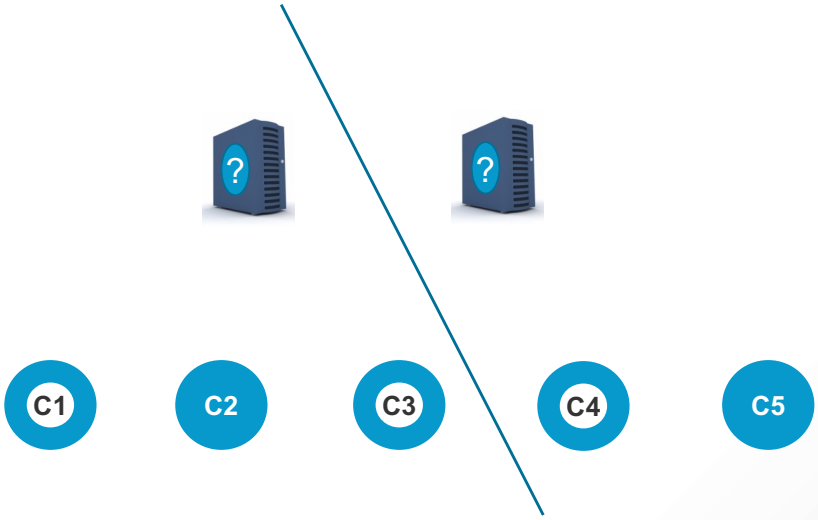


This picture depicts:

* A stripe with two servers, operating normally, with one active server, one mirror server,
* There are five clients running
* Each client has been configured to have an embedded voter service, but only three of them are registered and can cast votes (because the server configuration is failover-priority=consistency:3)
* If one of the clients currently registered to vote fails, one of the other clients will register themselves to vote

With two servers and three voters, elections will consist of a total of 5 expected votes, resulting in a total of 3 votes required for quorum (>50%).

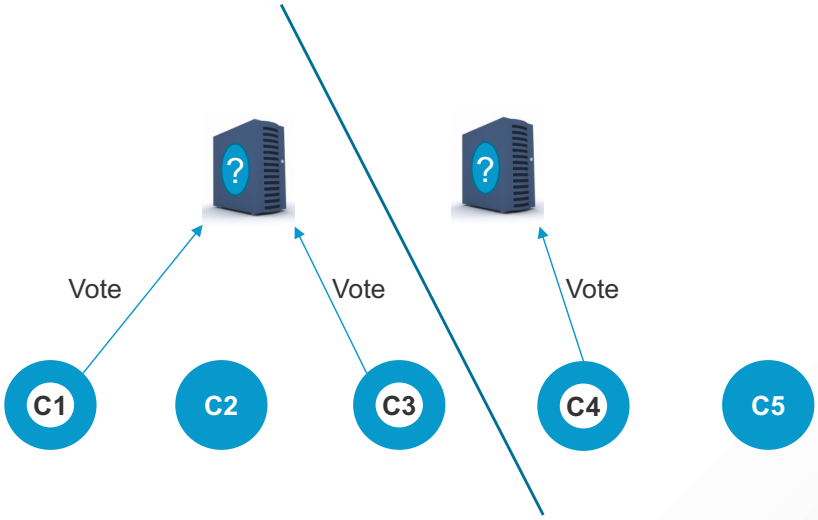
VOTER EMBEDDED CLIENTS



This picture depicts:

* A network partition has occurred
* One side of the partition contains: one server, one client that is an active voter, and one client that is not an active voter
* The other side of the partition contains: one server, two clients that are active voters, and one client that is not an active voter

VOTER EMBEDDED CLIENTS

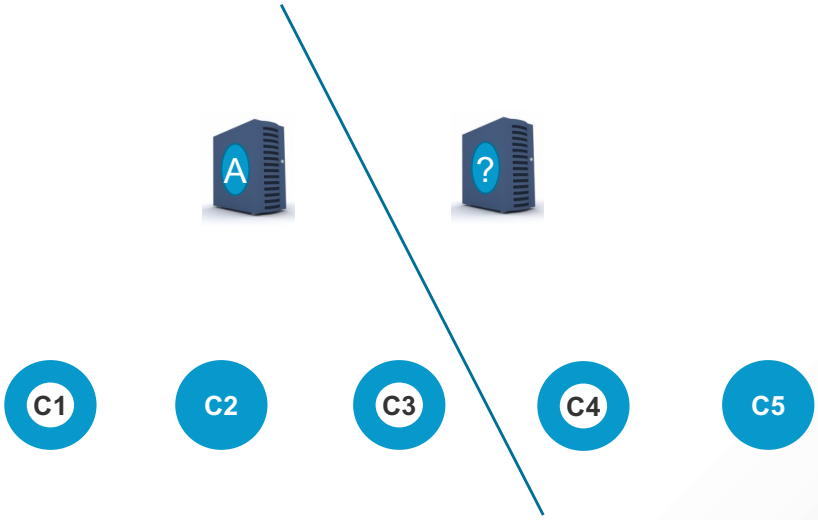


 |

This picture depicts:

* An election is held
* Each server casts a vote for itself
* One server can only receive one additional vote
* The other server can receive two additional votes

VOTER EMBEDDED CLIENTS



This picture depicts:

* The election completes
* The server that lost the election remains suspended, and the clients that can only see it are unable to perform work
* The server that received 3 total votes becomes active, and the clients that can see it resume work