SERVER REPLICATION

Replication may be used for improved reliability or improved efficiency. A major requirement is transparency.

Two approaches:

- active replication (with no centralised control). A voting mechanism masks failure.
- Primary-backup. Less redundancy in processing therefore used more in practice

Active Replication

Addresses 2 classes of faulty behaviour:

- Byzantine failure — faulty components exhibit arbitrary and malicious behaviour (possibly in collusion with other faulty components)
- failstop — on failure the component changes to a state that permits other components to detect its failure then it stops.

A component is run on every processor in the distributed system. If each one starts in the same state, is running on a non-faulty processor, and executes the **same requests** in the **same order**, they should do the same things and produce the same outputs.

This gives rise to the requirements:

- **agreement** — every non-faulty replica receives every request
- **order** — every non-faulty replica has the requests it receives in the same relative order.

Under some conditions these requirements can be weakened e.g.,

- agreement can be relaxed for read-only requests when failstop processors are assumed (stops processing and other processors can detect the failure). (For a non-update operation only one response is needed as long as it can be guaranteed to be correct. The request can be sent to a subset of processors and the failstop requirement can be used to exclude faulty ones.)
- order can be relaxed for operations that commute.

Agreement

Either the client or one of the replicas can perform the task of disseminating the request to all replicas. (e.g., using a reliable broadcast protocol to ensure all receive it)

Order

This can be achieved by assigning unique identifiers to requests — identifiers that have a global total ordering. e.g., based on a logical clock, synchronised realtime clocks, or by some sort of agreement between server replicas.

Voting:

- unanimous
- majority
- first come

Only non-faulty replicas are considered in the voting.

Identifying and Isolating Faulty Replicas

See Mullender section 7.8.1.

When a faulty processor is identified its output must no longer be used in the voting. Identifying faulty elements is possible (reliably) with failstop processors but isn't always possible where Byzantine failures can occur.

A **configurator** detects failure and recovery of elements in the system and may add or remove them subject to the restrictions:

- only faulty elements are removed
- only non-faulty elements are added

Network Partitioning

Network partitioning (rather than failure of a processor) may cause some elements to be inaccessible and unable to participate in the voting.

The response to this may be to:

- kill all replicas in the smaller partition
- allow processing to continue in some form in both partitions (e.g., posting news on the Internet, but this relies on the order of operations not being critical — they can be saved up and passed on to the other part of the network when it is reintegrated).

Reintegration

Faulty elements can only be reintegrated if they can be brought into an appropriate state (the same state as the other non-faulty elements are in at the point in time when is is to join).

- After a partial failure the failed element can copy the current state from some other correct element.
- On total failure restart from a checkpoint made while the element is in a stable state can be used. The checkpoint can be done after each transaction (very expensive) or periodically (and subsequent transactions saved in a log so that they can be replayed to restore the state).

  It is also possible to checkpoint a correct element, restart it and migrate a copy of the checkpoint image to the location where recovery is needed.

ISIS

ISIS enables reliable distributed computing to be implemented based on process groups and multicast for communicating between members of the process group.

It keeps a list of processes that are members of the group. A process leaves the group when it fails (terminates abnormally or fails to respond).

It has the ability to send relevant components of application state to a newly joining member of the group and ensure that state is consistent between group members from the point of joining onwards.

If a network is partitioned only the partition which has the majority of the group members is allowed to keep processing. The rest are killed.

A read-any, write-all (operational copies) is used when accessing replicated data.

Primary-Backup

- Clients make requests only to the server designated as the primary server. If this changes, clients have to be notified of the change.
- If the primary server fails one of the backups takes over (**failover**)
- No more than one server is primary at any one time. **Failover time** is the longest time during which no server is a primary server.
- If a client request arrives at a server that is not the current primary server it is lost.
- Only a bounded number of failures can be tolerated in any period during the lifetime of the system (where a period is bounded by the point at which repaired servers are reintegrated with the system.

There are different ways of handling failover.

A request can be lost during failover, therefore the protocol needs to be able to retry requests.

A Simple Protocol

A client sends a request to the primary server. When the primary server receives the request it:

- processes the request
- sends information about the update (a state update message) to the secondary server
- without waiting for an acknowledgement from the secondary server it sends a response to the client

Non update requests need only be seen by the primary server.

Primary-backup protocols are not limited to a single secondary server (as in this example).

More reliable protocols can be achieved by using a more reliable means of ensuring update on the secondary server (e.g., 2 phase commit).

The primary server sends a message to the secondary server at regular intervals to tell the secondary server that it is still alive.

If the secondary server does not hear from the primary server in a certain length of time:

- it becomes the primary server,
- sends a message to clients and
- from them on services client requests.

HA-NFS

There are two servers connected to logical disk volumes (that may consist of multiple actual disks). The disks are not replicated.

In the single network configuration (shown on next slide) only a single server failure can be tolerated. There is a dual network configuration that can tolerate a network failure as well. It does not tolerate disk failure.

Each server has two network interfaces (primary and secondary) and two IP addresses.

The primary interface is for normal operation and its secondary interface is for **impersonating** the other server if it fails.

Normal operation

The client sends requests to the appropriate primary server for the disk volumes it is interested in and the server performs the request without interacting with the other server.

Periodically the servers exchange “are you alive” messages and responses.

Failover

If one server fails to respond to the “are you alive” message after a certain number of tries and it is satisfied that the other server is not just running very slowly, it takes over management of the other servers disks.

The live server may have to do some processing to bring the failed server’s file system into a consistent state.

The server changes the IP address of its secondary adapter to that of the primary adapter of the failed server. It now receives requests for the failed server on its secondary adapter.

Re-Integration

A repaired server comes up with its primary interface turned off (so that it doesn’t conflict with the other server).

Through its secondary interface it sends a request to the running server which stops its impersonation and then notifies the recovered server allowing it to switch on its primary interface and proceed.