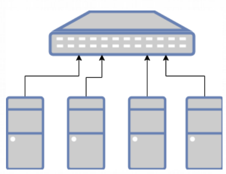
Quorum-less distributed writes

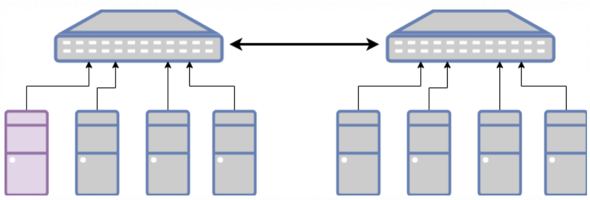
One of the most important issues that may arise in a distributed filesystem deployment is a critical condition called "split brain": a situation where a network is partitioned into one (or more) independent parts that cannot reach each other, and this causes two or more replicated copies of a file to become divergent. One of the ways in which this problem is prevented is through a write quorum – for a write to be acknowledged, it must be confirmed by a majority of nodes; this means that most systems can guarantee a reliable write only with a minimum of 3 nodes, and when degraded to two nodes they are unable to guarantee operations in case of further faults. NodeWeaver uses a different approach (called **version vectors**) – to guarantee reliable writes even in degraded configuration; this method minimizes the performance hit caused by quorum methods, focusing on the rare event (the split brain) instead of the common one (distributed writes). Version vectors allow to correct the damage caused by split brain events without interfering in standard IO operations on the cluster, by providing the information necessary to recover the correct information from the splitted nodes.

Why is split brain a problem?

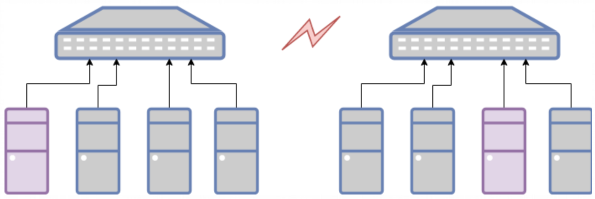
To understand it, we must first get an overview of how NodeWeaver handles the "normal" condition. A NodeWeaver cluster is composed of many servers, connected through a network; when the cluster becomes active, all the servers participate in a process called "master election", where one of the servers becomes the "master" and handles the coordination of the others. This coordination is loose, since in our platform the master does not directly control every aspect but sends messages through lightweight agents (called "remotes") that run on the individual nodes; the master must however be unique, to prevent two separate entities giving conflicting commands. So, the normal operation is like this:



With all the nodes connected to a single switch. If a node fails, the others are unable to reach its internal IP address, and the system decides after a predetermined period to mark it as "dead" and the data and VM that were hosted there are recreated on another available node. Now, let's imagine a more complex scenario:



We have now two switches, with an uplink between the two (the purple node is the master). Let's imagine that the uplink fails, so that we end up with two separate networks; in this condition, the second network is unable to reach the old master (and the other 3 nodes) and will thus miss a master. This will force a new election, and each network partition will assume the other nodes disappeared, forcing a replication of all the data and VMs that were on the other side:



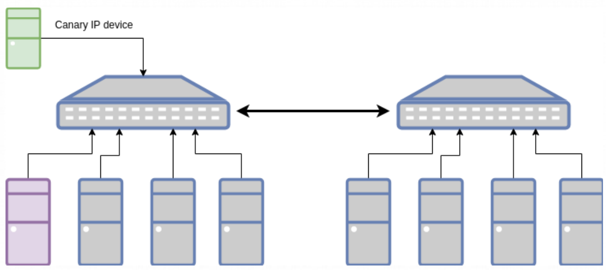
This situation is known as **split brain**; a condition where two sides are not able to communicate but continue to work independently. The problem is what happens when the link is reconnected - you end up with two masters, two copies of all data and VMs that may have diverging contents.

Canary IP protection

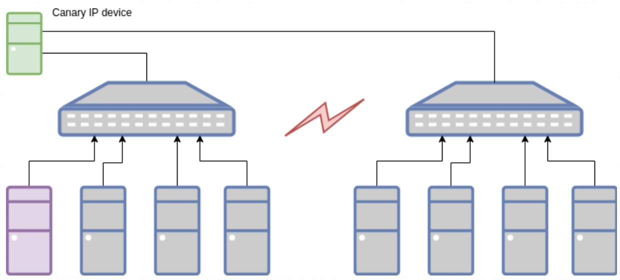
To prevent this, NodeWeaver uses an external device that is reachable with ICMP pings, called a Canary IP. The canary IP must be external to the cluster, and should be reachable at all times when the network works properly, but unreachable when the network fails. It may be a router, a physical server or desktop, even a small, dedicated device like a preconfigured Raspberry-PI device that is left always on on the network. It must not be a VM running on the same NodeWeaver cluster, as it would provide obviously no protection at all against its own split condition.

Quorum-less distributed writes

With the Canary IP device connected to the network, the resulting design appears like this:



What happens now if the link between the switches goes down? The nodes that are connected to the switch on the right are unable to reach the canary IP, so they assume that the network is not working properly, and thus will not cause an election - preventing split brain. IO will be suspended, and after a programmable delay the VMs will be shut down (and assuming they are configured to do so, they will be restarted on the nodes on the left). The Canary IP device does not need any special software or configuration, and apart from responding to ICMP pings it will not need any additional functionality (the IP **must not be the management IP of one of the switches**, though - during a switch reboot, it may respond to pings but not route any packets!) There may be, however, configuration errors that lead to split brains - for example, a software partitioning; complex routing infrastructure may lead to something like this:



In this case, the canary responds to pings from both sides - and we return to a split brain condition! While much less frequent, this may still happen. What the nodes see is a sort of "parallel universe" - each side continues on different courses, starting from the point in time when the split happened. The real problem happens when the two sides get connected back again - and you have two copies of everything, but with different data and conflicting versions.

Version vectors to the rescue

In this situation, NodeWeaver recovers gracefully thanks to our **version vectors**; to understand how this works, we first have to explain a bit about how NodeWeaver saves data.

Everything within NodeWeaver is stored in a distributed filesystem that breaks data into small blocks of variable size (from 64Kb up to 64MB) called "chunks". Each chunk includes not only the data, but also a CRC verification and a "version vector", that gets updated at every write. In a sense, we always create new chunks that silently replace the old ones; this is necessary because we don't confirm a write to have happened until as many nodes as the replication level required (the "goal") have confirmed the local write.

So, to return to our split brain event: the two sides of the network elect their master, VMs get replicated and data is copied again. What happens when the two sides reconnect? An internal NodeWeaver probe checks for the presence of two masters, and the following happens:

- The master that is running for the longest time sends a shutdown message to the other, which is demoted to being a normal node
- The VMs that are replicated on the wrong side are deleted
- The (now lone) master starts checking the chunks for incongruences; all chunks that are part of the wrong "temporal line" are deleted and replaced with the correct ones.

We can do that, because the master **keeps a running log of all changes to the distributed file system**, keeping track of the CRC and version of each change; this way, when we encounter the chunks from the right side of the network the master can recognize those as inconsistent (because their CRC and version don't match with the running log) and replace them with the correct ones. This "running log" is called the metadata stream, and is sent continuously to all the nodes, so independently of which node resumes operations first, it will always be consistent.

The end result is that the coherence of the file system is preserved; this is however a quite complex and resource-hungry task - so plan in advance to prevent it from happening! This is the reason why NodeWeaver can work reliably with two nodes only - writes are not dependent on a "quorum", that is, a majority decision when executing writes; all changes to the distributed file system are treated like a transaction, and the transactions are "signed" through the CRC and version vector, so that a parallel cluster that split from the main one will have unreplayable transactions that are properly managed on cluster rejoin.