Convergent Distributed Storage with DRBD

Pierre-Philipp Braun <pbraun@nethence.com>

Sep 2022 (draft v0.3)

Introduction

DRBD version 9 provides the ability to reach the mirrored volumes, not only from the members of the replication itself but across all the nodes of a storage farm, thanks to the **diskless feature**. When combined with a farm of Virtual Machine Monitors (VMM) *aka* hypervisors, containers or anything else on the very same machines, DRBD v9 is also and *de-facto* ready for that, and allows to run e.g. XEN guests on any node beyond the mirror itself. The upper cluster farm lives on the same nodes and can be de-correleated from the storage layer. The disadvantage of this **fully convergent** setup is that we have to stick with DRBD v9 *protocol C*, which is less optimal than *protocol A* in terms of pure storage performance.

We are here covering various aspects of storage and cluster system farms:

- how the multiple mirrors are placed across the farm (the architecture with its super-simple distributed system algorithm)
- the need for *some* mecanism to reference the LUNs to be used by an orchestrator on the convergent layer from above
- the aim to favor volatile vs. non-volatile resource performances resp. promote a fully vs. partially convergent farm.

Chain-line of mirrors

For brevety and the purpose of a PoC, here's a cluster with only three nodes. It is, however, horizontally scalable. Every node has three disks or volumes, with or without hardware RAID, depending on how resilient the customer wants his data. The first disk is small and used for the system alone. The two other disks are larger and are used by DRBD9 to setup many mirrors on top of **LVM2 thin-provisioning**. We're talking about a *chain-line of multiple mirrors* here. First node replicates its LVM2 logical volumes from disk2 to the next node's disk3 and so forth. When ever a new node joins the farm, we need to temporary break of the leaf mirror and extend the chain. First, let us visualize how the mirrors are organized.

```
                                                    (horizontally scalable)

      +-------------+         +-------------+         +-------------+
      |    node1    |         |    node2    |         |    node3    |
      +-------------+         +-------------+         +-------------+
      |   nvme0n1   |         |   nvme0n1   |         |   nvme0n1   |
      +-------------+         +-------------+         +-------------+
      |   nvme1n1   +---+     |   nvme1n1   +---+     |   nvme1n1   +---+
      +-------------+   |     +-------------+   |     +-------------+   |
+---> |   nvme2n1   |   +---> |   nvme2n1   |   +---> |   nvme2n1   |   |
|     +-------------+         +-------------+         +-------------+   |
|                                                                       |
+-----------------------------------------------------------------------+
```

Second, let us emphasis on the fact that every node can reach any replicated DRBD volume using the wonderful DRBD v9 **diskless feature**. It's worth noting that FreeBSD HAST does *not* have this feature available therefore cannot deliver fully convergent farms out-of-the-box.

```
+------------------+        +------------------+        +------------------+
|      node1       |        |      node2       |        |      node3       |
|                  |        |                  |        |                  |
|    nvme1n1       |        |    nvme1n1       |        |    nvme1n1       |
| drbd101 local    |        | drbd201 local    |        | drbd301 local    |
| drbd102 local    |        | drbd202 local    |        | drbd302 local    |
| ...              |        | ...              |        | ...              |
|                  |        |                  |        |                  |
|    nvme2n1       |        |    nvme2n1       |        |    nvme2n1       |
| drbd301 mirror   |        | drbd101 mirror   |        | drbd201 mirror   |
| drbd302 mirror   |        | drbd102 mirror   |        | drbd202 mirror   |
| ...              |        | ...              |        | ...              |
|                  |        |                  |        |                  |
| drbd201 diskless |        | drbd301 diskless |        | drbd101 diskless |
| drbd202 diskless |        | drbd302 diskless |        | drbd102 diskless |
| ...              |        | ...              |        | ...              |
+------------------+        +------------------+        +------------------+
```

No Hype Required

no need for a storage orchestrator!

The Linstor product acts as an orchestrator to define and provision LUNs in a DRBD farm, and possibly by "plugging" it with other farm UIs and orchestrators, for example Proxmox (an UI for KVM and LXC) and XCP-NG (an UI and orchestrator for XEN hypervisors). It is our thesis that for specific cases and very demanding customers, the Linstor product does not necessarily fit, as we can provision DRBD LUNs by following our scheme from above and integrate simple scripts and bypass the additional orchestrator altogether.

For example, a massive public-cloud provider would most probably have its own management and orchestration system, with no officially available Linstor plug-in for it. Yet if willing to switch to Linbit's product line for its storage, this customer would need to point to some DRBD LUNs by means of *some* mechanism, which would either involve using Linstor without a plug-in, or handling the management of the referenced LUNs within. We are attempting to promote the latter in this document.

Let us assume a virtualization farm. Its orchestrator already takes care of creating new instances on the nodes, based on CPU and RAM constrains. If we were to consider plugging our existing and in-house farm engine to DRBD storage nodes, be it convergent or remote diskless initiator nodes, here's three scenario to handle the storage farm directly:

- LUNs are provisioned statically and before-hand
- LUNs are provisioned dynamically based on LVM2 sparse/thin pool usage
- LUNs are provisioned dynamically based on I/O load monitoring

Those three scenario are described as follows.

Static LUNs

The DRBD volumes not need to be provisioned on-the-fly. We can define them in advance, in quantity and with some expected over-allocation. When the physical disks truly become busy (in terms of space e.g. 70%) and thin provioning comes to an end, the left-over drbd devices will simply be left as spares.

The cloud engine simply needs to pick-up DRBD node pairs in sequence to create new replicated volumes. Therefore all the nodes end-up with a progressively equal number of volumes and more approximately in terms of workload.

In case a new (leaf) node gets added, however, the cloud engine has to create all the new replicated volumes on it as for the primary target, and with its mirror on the first node). Once the leaf node acquires as much volumes as the other nodes, we're back to the strict round-robin allocation pattern.

In short, this setup aims to share an equal number of DRBD volumes on every node.

Dynamic LUNs

We are here considering dynamic allocation of new mirrors depending on physical space usage. We are looking for the number of available physical extends in every storage node’s thin-provisioned LVM2 volume group pool This information is easy to be gathered by SSH, possibly by SNMP or whatever means.

In short, the built-in script (instead of a “plug-in”) would be able to define which node is the least used in terms of *truly-used* disk space.

The question of thin-provisioning necessarily raises some concerns and we would need a procedure to handle the critical scenario where the storage needs of a node grows with time and possibly exceeds its originally thin-provisioned volumes. In other words, we need to be able to re-allocate a few volumes and liberate some space (and why not consider load also) on that to-be-overloaded node. And that should happen before it’s too late, obviously, which leads us to incident monitoring (for example an altert when ever less than 30% physical extends are available). But the next point is rather about *performance* monitoring.

Monitored LUNs

We are here considering dynamic allocation of new mirrors depending on disk I/O peformance. Monitoring systems such as Zabbix, Nagios XI, M/Monit or Prometheus could be very handy in that regard, as the data on disk load is already at hand (we can address their respective databases directly and read-only). Those performance monitoring systems inform us how much a given node is already suffering in terms of disk I/O per day (24 hours average).

In short, depending on the monitored disk I/O load on every DRBD node, the built-in script would be able to define which node has the most disk I/O resource available. This can also be coupled with the above, in case of thin-provisioning.

Conclusion & Demo

We at Nethence Systems stand available for a demo of the three-node PoC clusted described above (DRBD & XEN), and some draft of an in-house and fully convergent farm management engine.