Demonstrating the Capabilities of a Fault-Tolerant NVMe Solution:

Xinnor xiRAID Classic Software RAID and AIC HA202-PV Storage Bridge Bay Platform

This document showcases how xiRAID Classic software RAID and the AIC Storage Bridge Bay (SBB) platform can be combined to create a high-performance, fault-tolerant solution for storage subsystems. Our comprehensive testing demonstrates not only near-backend performance but also robust High Availability (HA) capabilities, making this solution ideal for complex high-performance computing environments and other markets utilizing parallel file systems such as BeeGFS and Lustre.

About xiRAID Classic

xiRAID Classic is a high-performance software RAID solution specifically designed for NVMe storage devices and modern SAN networks. xiRAID Classic technology fully leverages the capabilities of Flash devices (NVMe, SAS, SATA) to create a fast, fault-tolerant RAID that functions as a local block device ready for network export via auxiliary software.

The idea behind implementing xiRAID Classic in High Availability mode is to present the RAID as a new type of cluster resource. Since xiRAID Classic naturally exposes a specific system resource (the storage block device), it is an ideal candidate for cluster resource management. Starting with version v4.1.0, xiRAID Classic includes a Pacemaker resource agent in the distribution to enable Pacemaker integration. In this configuration, xiRAID Classic ensures data protection on the underlying drives through RAID technology, while the cluster eliminates the single point of failure represented by the xiRAID Clssic software instance.

About AIC HA202-PV

The HA202-PV is a SBB (Storage Bridge Bay) platform featuring a shared storage backplane for two otherwise independent compute nodes in a single chassis. Both nodes have dedicated PCI-e x2 links to each NVMe drive. Connectivity details are provided below. This architecture enables greater speed and efficiency in storage backend utilization while providing redundant storage within the same footprint as traditional unprotected single-node solutions.

Test Approaches

Our testing strategy was divided into two main parts to provide a full-scale view of the solution's capabilities:

1. **Performance Comparison.** We conducted a detailed comparison of xiRAID Classic performance against raw drive performance. This involved establishing a baseline for maximum hardware performance, then creating and testing four RAID 6 arrays (8 data drives + 2 parity) to determine RAID efficiency. We used the AIC HA202-PV SBB platform, featuring 20 NVMe drives (HUSMR7616BDP301, 1.6 TB each) split into 40 namespaces for optimal throughput. Tests included sequential read/write operations and random read/write operations under various configurations.
2. **High Availability Configuration.** We set up and tested a fully functional HA cluster using xiRAID and AIC HA202-PV platform. This solution involved configuring a two-node cluster using Pacemaker and Corosync, implementing IPMI-based fencing, and setting up Csync2 for configuration synchronization. We created four RAID 6 arrays distributed across both nodes and configured them as cluster resources. The HA setup was rigorously tested through failover scenarios, failback operations, and fencing functionality to ensure robust fault tolerance and seamless operation in case of node failures.

The testing environment consisted of:

- **Platform:** AIC HA202-PV SBB
- **CPUs per node:** 2xIntel(R) Xeon(R) Gold 6230
- **RAM per node:** 256GB
- **Drives Subsystem:** 20xHUSMR7616BDP301 (1.6 TB)
- **Drive BUS:** PCI-e 3.0
- **Operating system:** Rocky Linux 9.4
- **xiRAID Classic version:** 4.1.0

RAIDs

name	static	state	devices	health	wear	serials	params	info
media1	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme8n1 online 1 /dev/nvme9n1 online 2 /dev/nvme4n1 online 3 /dev/nvme5n1 online 4 /dev/nvme6n1 online 5 /dev/nvme7n1 online 6 /dev/nvme0n1 online 7 /dev/nvme1n1 online 8 /dev/nvme2n1 online 9 /dev/nvme3n1 online	100% 100% 100% 100% 100% 100% 100% 100% 100% 100%	18% 21% 18% 19% 15% 19% 19% 20% 16% 19%	SDM00000D759_1 SDM000010538_1 SDM00000D7AC_1 SDM00000CFEE_1 SDM0000104F6_1 SDM00000D76A_1 SDM000010539_1 SDM00000D760_1 SDM00000D762_1 SDM00000CF42_1	init_prio : 100 recon_prio : 100 memory_limit_mb : 0 merge_read_enabled : 0 merge_write_enabled : 0 merge_read_wait_usecs : 300 merge_read_max_usecs : 1000 merge_write_wait_usecs: 300 merge_write_max_usecs : 1000 resync_enabled : 1 sched_enabled : 0 request_limit : 0 restripe_prio : 100 cpu_allowed : 20-39,60-79 adaptive_merge : False	memory_usage_mb : -
media2	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme14n1 online 1 /dev/nvme15n1 online 2 /dev/nvme16n1 online 3 /dev/nvme17n1 online 4 /dev/nvme10n1 online 5 /dev/nvme11n1 online 6 /dev/nvme12n1 online 7 /dev/nvme13n1 online 8 /dev/nvme18n1 online 9 /dev/nvme19n1 online	100% 100% 100% 100% 100% 100% 100% 100% 100% 100%	17% 19% 17% 19% 14% 13% 20% 19% 19% 18%	SDM0000104C0_1 SDM000010507_1 SDM00000B359_1 SDM000010DD0_1 SDM00000CF8B_1 SDM00001050F_1 SDM00000D765_1 SDM00000D79D_1 SDM00001050E_1 SDM00000D776_1	init_prio : 100 recon_prio : 100 memory_limit_mb : 0 merge_read_enabled : 0 merge_write_enabled : 0 merge_read_wait_usecs : 300 merge_read_max_usecs : 1000 merge_write_wait_usecs: 300 merge_write_max_usecs : 1000 resync_enabled : 1 sched_enabled : 0 request_limit : 0 restripe_prio : 100 cpu_allowed : 0-19,40-59 adaptive_merge : False	memory_usage_mb : -

Node 1

RAIDs

name	static	state	devices	health	wear	serials	params	info
media3	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme8n2 online 1 /dev/nvme9n2 online 2 /dev/nvme4n2 online 3 /dev/nvme5n2 online 4 /dev/nvme6n2 online 5 /dev/nvme7n2 online 6 /dev/nvme0n2 online 7 /dev/nvme1n2 online 8 /dev/nvme2n2 online 9 /dev/nvme3n2 online	100% 100% 100% 100% 100% 100% 100% 100% 100% 100%	19% 13% 19% 19% 17% 15% 18% 21% 18% 17%	SDM000010DD0_2 SDM00001050F_2 SDM000010507_2 SDM00000CFEE_2 SDM00000B359_2 SDM0000104F6_2 SDM00000D759_2 SDM000010538_2 SDM00000D7AC_2 SDM0000104C0_2	init_prio : 100 recon_prio : 100 memory_limit_mb : 0 merge_read_enabled : 0 merge_write_enabled : 0 merge_read_wait_usecs : 300 merge_read_max_usecs : 1000 merge_write_wait_usecs: 300 merge_write_max_usecs : 1000 resync_enabled : 1 sched_enabled : 0 request_limit : 0 restripe_prio : 100 cpu_allowed : 20-39,60-79 adaptive_merge : False	memory_usage_mb : -
media4	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme14n2 online 1 /dev/nvme15n2 online 2 /dev/nvme16n2 online 3 /dev/nvme17n2 online 4 /dev/nvme10n2 online 5 /dev/nvme11n2 online 6 /dev/nvme12n2 online 7 /dev/nvme13n2 online 8 /dev/nvme18n2 online 9 /dev/nvme19n2 online	100% 100% 100% 100% 100% 100% 100% 100% 100% 100%	16% 14% 19% 20% 19% 19% 19% 20% 19% 18%	SDM00000D762_2 SDM00000CF8B_2 SDM00000D79D_2 SDM00000D765_2 SDM00000D76A_2 SDM000010539_2 SDM00000CF42_2 SDM00000D760_2 SDM00001050E_2 SDM00000D776_2	init_prio : 100 recon_prio : 100 memory_limit_mb : 0 merge_read_enabled : 0 merge_write_enabled : 0 merge_read_wait_usecs : 300 merge_read_max_usecs : 1000 merge_write_wait_usecs: 300 merge_write_max_usecs : 1000 resync_enabled : 1 sched_enabled : 0 request_limit : 0 restripe_prio : 100 cpu_allowed : 0-19,40-59 adaptive_merge : False	memory_usage_mb : -

Node 2

1. Comparing xiRAID Classic vs. Raw Drive Performance

Before starting any measurements, we split each NVMe (HUSMR7616BDP301) drive into two namespaces to achieve full throughput, resulting in 40 block devices. We also performed drive preconditioning before each test.

We first established a baseline as the maximum level of performance possible for the utilized hardware. Then, we created four RAID 6 arrays (8 data drives + 2 parity), ran tests on them, and compared the results to determine RAID efficiency. The FIO configuration files used for testing can be found in the appendix.

Full RAID initialization was performed on every array before testing. Arrays on each server node were bound to the NUMA node that the underlying drives were connected to. This was done to avoid expensive inter-CPU communication and resource allocation conflicts.

Detailed RAID configuration information is provided below. Comprehensive description of the system setup and the process of creating RAID arrays are provided in HA configuration part.

Test results

Raw 20 drives performance results

Test Scenario	**Combined result** **(2 nodes, 20 drives)**
RAW sequential read (numjobs=1, queue depth=32)	56.4 GB/s
RAW sequential write (numjobs=1, queue depth=32)	44.4 GB/s
RAW random read (numjobs=1, queue depth=64)	8.1 M IOPS
RAW mixed read 50% / write 50% (numjobs=4, queue depth=64)	5.5 M IOPS

Baseline performance

- Sequential read: 100% of raw backend performance (56.4 GB/s)
- Sequential write: 80% of raw backend performance (44.4 GB/s). Calculation: 44.4 GB/s x 80% = 35.5 GB/s
- Random read: 100% of raw backend performance (8.1M IOPS)
- Random write: We evaluated the performance of 20 NVMe drives (40 namespaces) under simultaneous load with a **50/50 r/w** pattern. The measured IOPS were 5.569k IOPS (**Read IOPS**: 2.783k; **Write IOPS**: 2.786k). Based on these measurements, the expected performance of 4 RAID 6 arrays can be calculated as:

$$IOPSraid = \frac{IOPS\ raw}{Write\ penalty} = \frac{5569k\ IOPS}{6} = 928k\ IOPS$$

xiRAID Performance Test Results

Test	**xiRAID Classic** (4 * RAID 6 (8d+2p))	**Baseline**	**Effic.**
RAID sequential write (numjobs=8, queue depth=32)	34.2 GB/s	35.5GB/s	96%
RAID sequential read (numjobs=8, queue depth=32)	55.4 GB/s	56.4GB/s	98%
RAID random write (numjobs=32, queue depth=32)	839k IOPS	928k IOPS	90%
RAID random read (numjobs=32, queue depth=32)	7.8 M IOPS	8.1 M IOPS	96%

2. HA Configuration Capabilities of xiRAID Classic and AIC's SBB Platform

xiRAID Classic can operate within a High Availability (HA) cluster, ensuring data integrity and availability for users by distributing data and system components across two nodes.

Pacemaker version	2.1.7
Corosync version	3.1.8
Csync^2 version	2.1
Fencing agent	fence-agents-ipmilan

Network configuration

Check that all IP addresses are resolvable from both hosts. In our case, we will use resolving via the hosts file, so we have the following content in `/etc/hosts` at both nodes:

```
127.0.0.1   localhost localhost.localdomain
localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain
localhost6 localhost6.localdomain6
10.10.0.11  node11
10.10.0.12  node12
```

Software components installation

xiRAID Classic 4.1 installation

Perform the following steps to install xiRAID Classic 4.1.0 on both nodes. Install EPEL.

```
# yum install -y epel-release
```

Install kernel-headers for the currently loaded kernel.

```
# yum install kernel-devel-$(uname -r)
```

Install xiraid-repo for current OS.

```
# yum install
https://pkg.xinnor.io/repository/Repository/x
iraid/el/9/kver-5.14/xiraid-repo-1.1.0-
446.kver.5.14.noarch.rpm
```

Install xiraid-release.

```
# yum install xiraid-release
```

Pacemaker installation

Running the following steps at both nodes: Enable cluster repo

```
# yum config-manager --set-enabled
highavailability crb
```

Installing cluster:

```
# yum install pcs pacemaker psmisc
policycoreutils-python3 fence-agents-ipmilan
```

Csync^2 installation

Install the Csync^2 package from the Xinnor repository at both nodes:

```
# yum install csync2
```

HA cluster setup

Pacemaker cluster creation

In this chapter, the cluster configuration is described. Set the firewall to allow pacemaker software to work (at both nodes):

```
# firewall-cmd --permanent --add-
service=high-availability
# firewall-cmd --reload
```

Set the same password for the hacluster user at both nodes:

```
# passwd hacluster
```

Start the cluster software at both nodes:

```
# systemctl start pcsd.service
# systemctl enable pcsd.service
```

Authenticate the cluster nodes from one node:

```
# pcs host auth node11 node12 -u hacluster
Password:
node01: Authorized
node02: Authorized
```

Create and start the cluster (start at one node):

```
# pcs cluster setup xraidha node11 node12
No addresses specified for host 'node11',
using 'node11'
No addresses specified for host 'node12',
using 'node12'
Destroying cluster on hosts: 'node11',
'node12'...
node12: Successfully destroyed cluster
node11: Successfully destroyed cluster
Requesting remove 'pcsd settings' from
'node11', 'node12'
node11: successful removal of the file 'pcsd
settings'
node12: successful removal of the file 'pcsd
settings'
Sending 'corosync authkey', 'pacemaker
authkey' to 'node11', 'node12'
node11: successful distribution of the file
'corosync authkey'
node11: successful distribution of the file
'pacemaker authkey'
node12: successful distribution of the file
'corosync authkey'
node12: successful distribution of the file
'pacemaker authkey'
Sending 'corosync.conf' to 'node11', 'node12'
node11: successful distribution of the file
'corosync.conf'
node12: successful distribution of the file
'corosync.conf'
Cluster has been successfully set up.
```

Start the configured cluster:

```
# pcs cluster start --all
node12: Starting Cluster...
node11: Starting Cluster...
```

Fencing setup

It's very important to have properly configured and working fencing (STONITH) in any HA cluster that works with shared storage devices. In our case, the shared devices are all the NVMe namespaces we created earlier. The fencing (STONITH) design should be developed and implemented by the cluster administrator in consideration of the system's abilities and architecture. In this system, we will use fencing via IPMI. Anyway, when designing and deploying your own cluster, please choose the fencing configuration on your own, considering all the possibilities, limitations, and risks.

We will use IPMI fencing agent, you may check the IPMI fencing agent options description by running the following command:

```
# pcs stonith describe fence_ipmilan
```

Adding the fencing resources:

```
# pcs stonith create node11.stonith
fence_ipmilan ip="10.10.0.21" auth=password
password="admin" username="admin"
method="onoff" lanplus=true
pcmk_host_list="node11"
pcmk_host_check=static-list op monitor
interval=10s
# pcs stonith create node12.stonith
fence_ipmilan ip="10.10.0.22" auth=password
password="admin" username="admin"
method="onoff" lanplus=true
pcmk_host_list="node12"
pcmk_host_check=static-list op monitor
interval=10s
```

Preventing the STONITH resources from start at the node it should kill:

```
# pcs constraint location node11.stonith
avoids node11=INFINITY
# pcs constraint location node12.stonith
avoids node12=INFINITY
```

Csync² configuration

Configure firewall to allow Csync² to work (run at both nodes):

```
# firewall-cmd --permanent --add-
port=30865/tcp
# firewall-cmd –reload
```

Create the Csync² configuration file /usr/local/etc/csync2.cfg with the following content at one node only:

```
# vi /usr/local/etc/csync2.cfg
nossl * *;
group csxiha {
 host node11;
 host node12;
 key /usr/local/etc/csync2.key_ha;
 include /etc/xiraid/raids;
}
```

Generate the key:

```
# csync2 -k /usr/local/etc/csync2.key_ha
```

Copy the config and the key file to the second node:

```
# scp /usr/local/etc/csync2.cfg
/usr/local/etc/csync2.key_ha
node12:/usr/local/etc/
```

For Csync² synchronization by schedule one time per minute run `crontab -e` at both nodes and add the following record:

```
# crontab -e
* * * * /usr/local/sbin/csync2 -x
```

Also, for asynchronous synchronization run the following command to create a synchronization script (repeat the script creation procedure at both nodes):

```
# vi /etc/xiraid/config_update_handler.sh
```

Fill the created script with the following content:

```
#!/usr/bin/bash
/usr/local/sbin/csync2 -xv
```

Save the file.

After that run the following command to set correct permissions for the script file:

```
# chmod +x
/etc/xiraid/config_update_handler.sh
```

xiRAID Classic Configuration for cluster setup

Disable RAID autostart to prevent RAIDs from being activated by xiRAID Classic itself during a node boot. In a cluster configuration, RAIDs have to be activated by Pacemaker via cluster resources. Run the following command at both nodes:

```
# xicli settings cluster modify --
raid_autostart 0
```

Make xiRAID Classic 4.1 resource agent visible for Pacemaker (run command this sequence at both nodes):

```
# mkdir -p /usr/lib/ocf/resource.d/xraid
# ln -s /etc/xraid/agents/raid
/usr/lib/ocf/resource.d/xraid/raid
```

xiRAID RAIDs creation

Creating all the RAIDs at the first node:

```
# xicli raid create -n media1 -l 6 -d
/dev/nvme8n1 /dev/nvme9n1 /dev/nvme4n1
/dev/nvme5n1 /dev/nvme6n1 /dev/nvme7n1
/dev/nvme0n1 /dev/nvme1n1 /dev/nvme2n1
/dev/nvme3n1 -ss 128

# xicli raid create -n media2 -l 6 -d
/dev/nvme14n1 /dev/nvme15n1 /dev/nvme16n1
/dev/nvme17n1 /dev/nvme10n1 /dev/nvme11n1
/dev/nvme12n1 /dev/nvme13n1 /dev/nvme18n1
/dev/nvme19n1 -ss 128

# xicli raid create -n media3 -l 6 -d
/dev/nvme8n2 /dev/nvme9n2 /dev/nvme4n2
/dev/nvme5n2 /dev/nvme6n2 /dev/nvme7n2
/dev/nvme0n2 /dev/nvme1n2 /dev/nvme2n2
/dev/nvme3n2 -ss 128

# xicli raid create -n media4 -l 6 -d
/dev/nvme14n2 /dev/nvme15n2 /dev/nvme16n2
/dev/nvme17n2 /dev/nvme10n2 /dev/nvme11n2
/dev/nvme12n2 /dev/nvme13n2 /dev/nvme18n2
/dev/nvme19n2 -ss 128
```

Since we already configured config replication, after this RAID config files should be replicated on another node. This can be verified with xicli raid show which will show the new RAIDs available but offline.

Also, let's create the filesystems on the newly created RAIDs.

```
# mkfs.ext4 /dev/xi_media1
# mkfs.ext4 /dev/xi_media2
# mkfs.ext4 /dev/xi_media3
# mkfs.ext4 /dev/xi_media4
```

Next, we need to prepare the mount points. Unlike RAIDs, filesystems and their mount points must be created on each of the cluster nodes separately.

```
# mkdir -p /mnt/media1
# mkdir -p /mnt/media2
# mkdir -p /mnt/media3
# mkdir -p /mnt/media4
```

Cluster resources creation

To create Pacemaker resources for xiRAID Classic RAIDs, we will use the xiRAID resource agent, which was installed with xiRAID Classic and made available to Pacemaker in one of the previous steps.

Unload all the RAIDs at the node where they are active:

```
# xicli raid unload --all
```

Getting the RAIDs UUIDs.

```
# grep uuid /etc/xiraid/raids/*.conf
/etc/xiraid/raids/media1.conf:    "uuid":
"763E66C5-B8E8-4206-8BCB-AD798770F8DD",
/etc/xiraid/raids/media2.conf:    "uuid":
"5C6C2470-9671-4DC8-B1D8-8FFFCF5B3D2D",
/etc/xiraid/raids/media3.conf:    "uuid":
"D113E2B0-5110-41D4-A926-CF5B1F6F90E7",
/etc/xiraid/raids/media4.conf:    "uuid":
"B9F28B55-8440-4311-B0FD-FA5BE8FC9E88",
```

Create pcs resource for RAIDs using UUIDs from previous step, each resource assign to its group (for example: `--group=gr_media1`).

```
# pcs resource create xi_media1
ocf:xraid:raid name=media1 uuid=763E66C5-
B8E8-4206-8BCB-AD798770F8DD op monitor
interval=5s meta migration-threshold=1 --
group=gr_media1

# pcs resource create xi_media2
ocf:xraid:raid name=media2 uuid=5C6C2470-
9671-4DC8-B1D8-8FFFCF5B3D2D op monitor
interval=5s meta migration-threshold=1 --
group=gr_media2

# pcs resource create xi_media3
ocf:xraid:raid name=media3 uuid=D113E2B0-
5110-41D4-A926-CF5B1F6F90E7 op monitor
interval=5s meta migration-threshold=1 --
group=gr_media3

# pcs resource create xi_media4
ocf:xraid:raid name=media4 uuid=B9F28B55-
8440-4311-B0FD-FA5BE8FC9E88 op monitor
interval=5s meta migration-threshold=1 --
group=gr_media4
```

and also, for the filesystems (notice that we assign them to the same groups with the RAID they have been created on):

```
# pcs resource create fs_media1 Filesystem
device="/dev/xi_media1"
directory="/mnt/media1" fstype="ext4" --
group=gr_media1

# pcs resource create fs_media2 Filesystem
device="/dev/xi_media2"
directory="/mnt/media2" fstype="ext4" --
group=gr_media2

# pcs resource create fs_media3 Filesystem
device="/dev/xi_media3"
directory="/mnt/media3" fstype="ext4" --
group=gr_media3

# pcs resource create fs_media4 Filesystem
device="/dev/xi_media4"
directory="/mnt/media4" fstype="ext4" --
group=gr_media4
```

Start order and location preferences

In xiRAID Classic 4.1, it is required to guarantee that only one RAID can be starting at a time. To do so, we define the following constraints. This limitation will be removed in future releases.

```
# pcs constraint order start xi_media1 then
start xi_media2 kind=Serialize
# pcs constraint order start xi_media1 then
start xi_media3 kind=Serialize
# pcs constraint order start xi_media1 then
start xi_media4 kind=Serialize
# pcs constraint order start xi_media2 then
start xi_media3 kind=Serialize
# pcs constraint order start xi_media2 then
start xi_media4 kind=Serialize
# pcs constraint order start xi_media3 then
start xi_media4 kind=Serialize
```

In order to run two resource sets on two different nodes, let's define the resource location preferences:

```
# pcs constraint location gr_media1 prefers
node11=50
# pcs constraint location gr_media2 prefers
node11=50
# pcs constraint location gr_media3 prefers
node12=50
# pcs constraint location gr_media4 prefers
node12=50
```

Verify cluster

Use pcs status to verify cluster and resources.

```
# pcs status
Cluster name: xraidha
Cluster Summary:
Stack: corosync (Pacemaker is running)
Current DC: node11 (version 2.1.7-5.2.el9_4-
0f7f88312) - partition with quorum
Last updated: Wed Oct  2 03:03:57 2024 on
node11
Last change:  Wed Oct  2 03:03:46 2024 by
root via root on node11
2 nodes configured
10 resource instances configured
Node List:
Online: [ node11 node12 ]
Full List of Resources:
node11.stonith     (stonith:fence_ipmilan):
Started node12
node12.stonith     (stonith:fence_ipmilan):
Started node11
Resource Group: gr_media1:
xi_media1    (ocf:xraid:raid):    Started
node11
fs_media1    (ocf:heartbeat:Filesystem):
Started node11
Resource Group: gr_media2:
xi_media2    (ocf:xraid:raid):    Started
node11
fs_media2    (ocf:heartbeat:Filesystem):
Started node11
Resource Group: gr_media3:
xi_media3    (ocf:xraid:raid):    Started
node12
fs_media3    (ocf:heartbeat:Filesystem):
Started node12
Resource Group: gr_media4:
xi_media4    (ocf:xraid:raid):    Started
node12
fs_media4    (ocf:heartbeat:Filesystem):
Started node12
Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Now we have a fully set up Pacemaker cluster with RAID and filesystem resources linked together and managed dynamically. If one node of the cluster fails, Pacemaker will start resources the other node. At the same time, if only a particular drive fails, xiRAID Classic will take care of it, going into degraded mode on the node where it is active, and failover to another node will not happen.

High Availability Tests

Failover

To test failover, we shutdown the second node (node12), and check the cluster status to confirm that all RAID arrays are active on the first node (node11):

```
# pcs status
Cluster name: xraidha
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: node11 (version 2.1.7-5.2.el9_4-0f7f88312) - partition with quorum
  * Last updated: Wed Oct  2 03:06:48 2024 on node11
  * Last change:  Wed Oct  2 03:03:46 2024 by root via root on node11
  * 2 nodes configured
  * 10 resource instances configured
Node List:
  * Online: [ node11 ]
  * OFFLINE: [ node12 ]
Full List of Resources:
  * node11.stonith  (stonith:fence_ipmilan):     Stopped
  * node12.stonith  (stonith:fence_ipmilan):     Started node11
  * Resource Group: gr_media1:
    * xi_media1 (ocf:xraid:raid):    Started node11
    * fs_media1 (ocf:heartbeat:Filesystem):  Started node11
  * Resource Group: gr_media2:
    * xi_media2 (ocf:xraid:raid):    Started node11
    * fs_media2 (ocf:heartbeat:Filesystem):  Started node11
  * Resource Group: gr_media3:
```

```
    * xi_media3 (ocf:xraid:raid):    Started node11
    * fs_media3 (ocf:heartbeat:Filesystem):  Started node11
  * Resource Group: gr_media4:
    * xi_media4 (ocf:xraid:raid):    Started node11
    * fs_media4 (ocf:heartbeat:Filesystem):  Started node11
Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled

# xicli raid show
```

RAIDs

name	static	state	devices	info
media1	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme15n1 online 1 /dev/nvme4n1 online 2 /dev/nvme16n1 online 3 /dev/nvme14n1 online 4 /dev/nvme2n1 online 5 /dev/nvme5n1 online 6 /dev/nvme7n1 online 7 /dev/nvme10n1 online 8 /dev/nvme12n1 online 9 /dev/nvme13n1 online	
media2	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme0n1 online 1 /dev/nvme3n1 online 2 /dev/nvme1n1 online 3 /dev/nvme6n1 online 4 /dev/nvme8n1 online 5 /dev/nvme9n1 online 6 /dev/nvme11n1 online 7 /dev/nvme17n1 online 8 /dev/nvme18n1 online 9 /dev/nvme19n1 online	
media3	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme15n2 online 1 /dev/nvme4n2 online 2 /dev/nvme16n2 online 3 /dev/nvme14n2 online 4 /dev/nvme2n2 online 5 /dev/nvme5n2 online 6 /dev/nvme7n2 online 7 /dev/nvme10n2 online 8 /dev/nvme12n2 online 9 /dev/nvme13n2 online	
media4	size: 5964 GiB level: 6 strip_size: 128 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme0n2 online 1 /dev/nvme3n2 online 2 /dev/nvme1n2 online 3 /dev/nvme6n2 online 4 /dev/nvme8n2 online 5 /dev/nvme9n2 online 6 /dev/nvme11n2 online 7 /dev/nvme17n2 online 8 /dev/nvme18n2 online 9 /dev/nvme19n2 online	

```
# df -h | grep media
/dev/xi_media1       5.8T   28K  5.5T   1% /mnt/media1
```

```
/dev/xi_media2       5.8T   28K  5.5T   1% /mnt/media2
/dev/xi_media3       5.8T   28K  5.5T   1% /mnt/media3
/dev/xi_media4       5.8T   28K  5.5T   1% /mnt/media4
```

As we see, all RAID arrays became available on the first node (node11), and all filesystems are mounted successfully.

Failback

After failover test, we turn the second node (node12) on and start the cluster, confirm that failover cluster resources failback:

```
# pcs status
Cluster name: xraidha
Cluster Summary:
  * Stack: corosync (Pacemaker is running)
  * Current DC: node11 (version 2.1.7-
5.2.el9_4-0f7f88312) - partition with quorum
  * Last updated: Wed Oct  2 03:11:02 2024 on
node12
  * Last change:  Wed Oct  2 03:03:46 2024 by
root via root on node11
  * 2 nodes configured
  * 10 resource instances configured
Node List:
  * Online: [ node11 node12 ]
Full List of Resources:
  * node11.stonith  (stonith:fence_ipmilan):
Started node12
  * node12.stonith  (stonith:fence_ipmilan):
Started node11
  * Resource Group: gr_media1:
    * xi_media1     (ocf:xraid:raid):
Started node11
    * fs_media1
      (ocf:heartbeat:Filesystem):
Started node11
  * Resource Group: gr_media2:
    * xi_media2     (ocf:xraid:raid):
Started node11
    * fs_media2
      (ocf:heartbeat:Filesystem):
Started node11
  * Resource Group: gr_media3:
    * xi_media3     (ocf:xraid:raid):
Started node12
    * fs_media3
      (ocf:heartbeat:Filesystem):
Started node12
  * Resource Group: gr_media4:
    * xi_media4     (ocf:xraid:raid):
Started node12
    * fs_media4
      (ocf:heartbeat:Filesystem):
Started node12
Daemon Status:
  corosync: active/disabled
  pacemaker: active/disabled
  pcsd: active/enabled
```

Fencing

Even in case when one node is unreponsive, we can't be sure it isn't accessing data. It is needed to fence the node using STONITH so we can be sure that the node goes offline, before allowing the data to be accessed from another node. In this case, the cluster uses STONITH to force the node offline by rebooting it.

To test the fence functionality, we disconnect the second node from the network and check the fence STONITH history:

```
# pcs stonith history
reboot of node12 successful: delegate=node11,
client=pacemaker-controld.4812,
origin=node11, completed='2024-10-02
22:04:44.456962 +08:00'
```

Also, we check that the second node was rebooted:

```
# last reboot | head -1
reboot   system boot  5.14.0-427.26.1. Wed
Oct  2 22:06   still running
```

Conclusion

The comprehensive testing of Xinnor xiRAID Classic with the AIC HA202-PV Storage Bridge Bay platform has demonstrated exceptional performance and efficiency across various workload patterns. Key findings include:

1. xiRAID Classic consistently achieved 90% to 98% of the baseline performance for both sequential and random operations. This negligible performance loss positions xiRAID Classic as one of the most efficient software RAID solutions available in the market.
2. The implemented HA cluster configuration demonstrated seamless failover and failback capabilities, ensuring continuous data accessibility even in the event of node failures.

These results underscore the viability of this software-defined storage solution for high-performance computing environments. The successful implementation and testing of the HA configuration further demonstrates the solution's readiness for production environments where downtime is not an option. The seamless failover and failback capabilities, combined with effective fencing mechanisms, ensure that the system can handle various failure scenarios without compromising data availability or integrity.

The combination of xiRAID Classic and AIC HA202-PV offers a cost-effective and flexible alternative to hardware RAID systems, particularly beneficial for parallel file systems like BeeGFS and Lustre. Its ability to deliver near-backend performance with fault tolerance makes it well-suited for data-intensive enterprise applications and research environments, where data integrity and access speed are vital, while also providing adaptability to evolving storage needs.

Appendix

Backend measurements job definitions

Precondition config for sequential tests

```
[global]
rw=write
bs=128K
iodepth=64
direct=1
ioengine=libaio
group_reporting
loops=2

[job 1]
filename=/dev/nvme0n1
[job 2]
filename=/dev/nvme0n2
...
```

Precondition config for random tests

```
[global]
rw=randwrite
bs=4K
iodepth=32
Numjobs=4
direct=1
ioengine=libaio
group_reporting
loops=2

[job 1]
filename=/dev/nvme0n1
[job 2]
filename=/dev/nvme0n2
...
```

RAW sequential write/read

```
[global]
rw=write
#rw=read
bs=128K
iodepth=32
direct=1
ioengine=libaio
runtime=100
group_reporting

[job 1]
filename=/dev/nvme5n1
...
[job 11]
filename=/dev/nvme0n1
...
```

RAW random read

```
[global]
rw=randread
bs=4K
iodepth=64
direct=1
ioengine=libaio
runtime=100
group_reporting

[job 1]
filename=/dev/nvme5n1
...
[job 11]
filename=/dev/nvme0n1
...
```

Mixed 50/50 random read/write

```
[global]
rwmixwrite=50
rw=randrw
numjobs=4
bs=4K
iodepth=64
direct=1
ioengine=libaio
runtime=100
group_reporting

[job 1]
filename=/dev/nvme5n1
...
[job 11]
filename=/dev/nvme0n1
...
```

RAID measurements job definitions

RAID sequential write/read

```
[global]
```

```
rw=write
#rw=read
bs=1M
iodepth=32
direct=1
ioengine=libaio
runtime=100
numjobs=8
offset_increment=12%
group_reporting

[job 1]
filename=/dev/xi_media1(3)
numa_cpu_nodes=1
[job 2]
filename=/dev/xi_media2 (4)
numa_cpu_nodes=0
```

RAID random read/write

```
[global]
rw=randwrite
#rw=randread
bs=4K
iodepth=32
direct=1
ioengine=libaio
runtime=100
numjobs=32
group_reporting

[job 1]
filename=/dev/xi_media1(3)
numa_cpu_nodes=1
[job 2]
filename=/dev/xi_media2(4)
numa_cpu_nodes=0
```