Western Digital.

White Paper

A High Performance BeeGFS Storage Solution powered by Western Digital's OpenFlex™ Data24 NVMe-oF™ Storage Platform and Xinnor xiRAID

Abstract

This white paper describes a solution for high-performance, high-throughput scalable storage on Western Digital's OpenFlex Data24 with BeeGFS and Xinnor software based xiRAID. This white paper explains the deployment, design, tuning guidelines, and observed performance.

February 2025

Table of Contents

Executive Summary

In high-performance computing (HPC), designing a well-balanced storage system to achieve optimal performance presents significant challenges. The HPC environment (which can consist of hundreds of nodes) processes high volumes of data to solve complex calculations. The HPC environment is such that it must be able to accommodate both large blocks of data for Big Data analytics and many small files for Machine Learning (ML) and Artificial Intelligence (AI) workloads without performance degradation. Organizations seek low-cost, high-performance, high-availability solutions that are also easy to manage whilst scaling to the growing demands of different types of IO-intensive workload challenges.

Composable Disaggregated Infrastructure (CDI) represents the modern architectural approach to data centre infrastructure, disaggregating compute, storage, and network resources into shared pools that can be composed for on-demand allocation.

The Western Digital OpenFlex Data24 NVMe-oF Storage Platform is a vertically integrated storage architecture that leverages an Open Composable Infrastructure (OCI) approach in the form of disaggregated data storage using NVMe-over-Fabrics (NVMe-oF).

xiRAID is a lightweight software RAID offering which complements CDI solutions whilst performing very closely to raw device capabilities and exceeding that of traditional hardware RAID solutions.

BeeGFS is an easily deployable hardware-independent POSIX parallel file system developed with a strong focus on performance whilst designed for ease of use, simple installation, and easy management.

The purpose of this document is to showcase a combined solution of BeeGFS cluster deployment with xiRAID volumes deployed on OpenFlex Data24 namespaces. This document is not an endorsement of xiRAID or BeeGFS by Western Digital, and no warranty of either product is either expressed or implied.

Problem Statement

HPC environments have been served by magnetic disks as a storage backbone for decades. Many applications in the HPC environment now must accommodate both large blocks of data and many small files. For instance, training an ML model might use millions of small files, while a big data analytics workload runs on one massive dataset. There is a need for increased metadata performance in many workloads today. Key components of the HPC cluster, such as compute, storage, and network, need to be evolved to serve these modern workloads. Many distributed parallel file systems have started to support the latest compute and NVMe-oF storage to utilize high performance NVMe™ SSDs effectively in HPC clusters. BeeGFS combines multiple storage servers to provide a highly scalable shared network file system using striped file contents, with which the high throughput demands of large numbers of clients can easily be satisfied.

Modern IO-intensive workloads like Big Data, ML, AI, and Internet of Things (IoT) workloads require data infrastructure designed to scale storage and compute independently to ensure both are provisioned efficiently and effectively. CDI ensures compute, storage, and network resources are placed into shared pools that can be composed for on-demand allocation. This enables compute to become stateless, elastic, and scalable independent of storage. Data durability and availability need to be ensured for an organization to provide continued operations or services. xiRAID is a high-performance software RAID that can be used to provide continued availability and fault tolerance for the storage.

In this document we demonstrate a solution that addresses the growing demand for high-performance parallel file systems for HPC clusters by deploying a high-performance distributed parallel file system on a composable infrastructure. Western Digital's validated design for HPC BeeGFS High-Capacity Storage solution with xiRAID is a fully supported, easy-to-use, high-throughput, scale-out, parallel file system storage solution with well-described performance characteristics.

Solution Highlights

The following sections of this paper provide an overview of the solution.

The solution combines:

- Western Digital's OpenFlex Data24 NVMe-oF Storage Platform. It provides low latency sharing of NVMe SSDs over a high-performance Ethernet fabric to deliver similar performance to locally attached NVMe SSDs.
- xiRAID is a high-performance software RAID developed specifically for NVMe storage devices to utilize up to 97% of hardware performance capabilities.
- BeeGFS is a high-performance parallel file system designed for performance-oriented environments like HPC, AI, and deep learning workloads. BeeGFS includes a distributed metadata architecture for scalability and flexibility reasons. Its most important aspect is data throughput.

By combining BeeGFS and xiRAID with Western Digital OpenFlex Data24, organizations can benefit from the parallel file systems:

- High Performance
- High Availability
- Scalability
- Robustness
- Easy to deploy and integrate with existing infrastructure
- Easy data management
- Optimized for highly concurrent access
- BeeGFS - software with enterprise features

Technology Overview

OpenFlex Data24 Overview

OpenFlex is Western Digital's architecture that supports OCI through storage disaggregation. The OpenFlex Data24 is a 2U rack-mounted data storage enclosure, built on the OpenFlex platform. This Just-a-Bunch-Of-Flash (JBOF) platform leverages the OCI approach in the form of disaggregated data storage using NVMe-oF. NVMe-oF is a networked storage protocol that allows storage to be disaggregated from compute, in turn makes that storage widely available to multiple applications and hosts. For more details, refer to OpenFlex Data24 NVMe-oF Storage Platform.

Enabling applications to share a common pool of storage capacity allows data to be easily allocated and / or shared between applications whilst independent of location. Exploiting NVMe device-level performance, NVMe-oF promises to deliver the lowest end-to-end latency from application to shared storage. NVMe-oF enables composable infrastructures to deliver the data locality benefits of NVMe DAS (low latency, high performance), while providing the agility and flexibility of sharing storage and compute.

Western Digital RapidFlex™ NVMe-oF fabric adapters are used to share storage to servers using the NVMe-oF protocol, either in direct connectivity or switched topology. OpenFlex Data24 is a vertically integrated Western Digital design with NVMe SSDs, fabric adapters, and a JBOF platform architecture. Each OpenFlex Data24 chassis is scalable up to 368TB[1] of low-latency dual-port SSDs in 2U 24-bay platform.

Composable Infrastructure seeks to disaggregate compute, storage, and networking fabric resources into shared resource pools that can be available for on-demand allocation (i.e., "composable"). Composability occurs at the software level, while disaggregation occurs at the hardware level using NVMe-oF. NVMe-oF will vastly improve compute and storage utilization, performance, and agility in the data center.

Western Digital's vision for OCI is based on the four key pillars below.

Open

- Open in both API and form factor.
- Designed for robust interoperability of multi-vendor solutions.

Scalable

- Delivering the ability to compose solutions at the width of the network.
- Enable self-organizing systems of composable elements that communicate horizontally.

Disaggregated

- Pools of resources available for any use case that is defined at run time.
- Independent scaling of compute and storage elements to maximize efficiency and agility.

Extensible

- Inclusive of both disk and flash.
- Entire ecosystem of composable elements managed and orchestrated using a common API framework.
- Prepared for yet-to-come composable elements – example: memory, accelerators.

Open Composable API – Western Digital's new Open Composable API is designed for data centre composability. It builds upon existing industry standards utilizing the best features of those standards as well as practices from proprietary management protocols.

OpenFlex is Western Digital's architecture that supports OCI through storage disaggregation – both disk and flash natively attached to a scalable fabric. OpenFlex does not rule out multiple fabrics, but whenever possible, ethernet will be used as a unifying connect for both flash and disk because of its broad applicability and availability.

[1] One terabyte (TB) is equal to one trillion bytes and one petabyte (PB) is equal to 1,000 TB. Actual user capacity may be less due to operating environment.

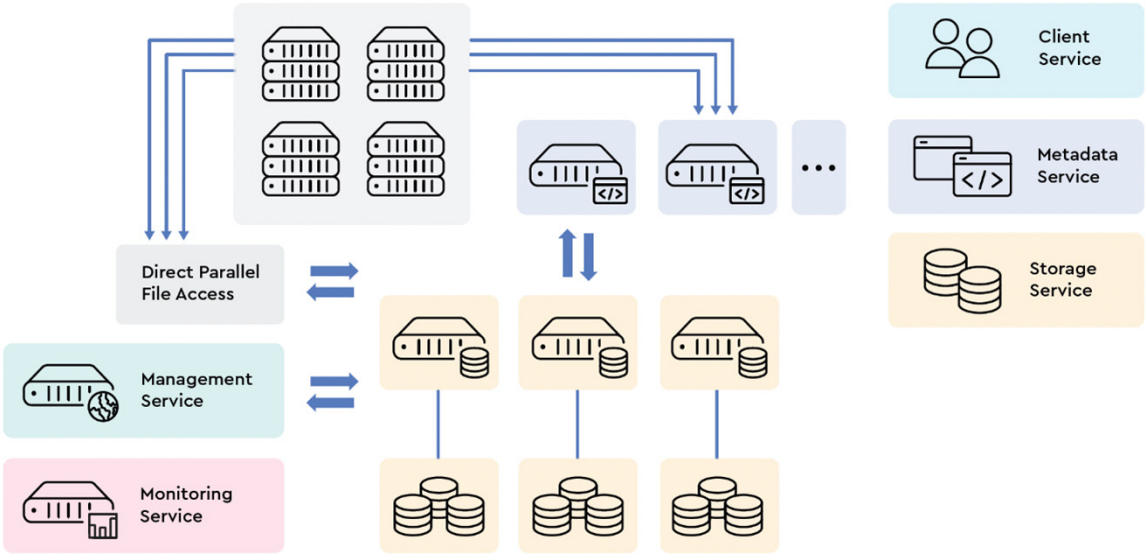
BeeGFS Overview

BeeGFS is a hardware-independent POSIX parallel file system (a.k.a., software-defined parallel storage) developed with a strong focus on performance and designed for ease of use, simple installation and easy management. It is designed for all performance-oriented environments, including HPC, AI, deep learning, life sciences, oil, gas, media, and entertainment.

One of the most fundamental concepts of BeeGFS is the strict avoidance of architectural bottlenecks or locking situations in the cluster through the user space architecture. BeeGFS stripes the file content on multiple storage nodes, plus it distributes file system metadata across multiple metadata servers, which benefits both small and large systems as well as metadata-intensive applications.

BeeGFS is built on highly efficient and scalable multithreaded core components with native RDMA support. File system nodes can serve RDMA (InfiniBand, Omni-Path, RoCE, and TCP/IP) network connections at the same time and automatically switch to a redundant connection path in case any of them fail.

Learn more about downloading and using BeeGFS along with its end user license agreement at www.beegfs.io.



The BeeGFS architecture is composed of four main services:

Management service

- The management service can be figured as a “meeting point” for the BeeGFS metadata, storage, and client services.
- Very light-weight and typically not running on a dedicated machine, as it is not critical for performance and stores no user data.
- Watches all registered services and checks their state and maintains it.
- This is the first service to be configured in a newly deployed environment.

Metadata service

- The metadata service stores information about the data, such as directory information, file and directory ownership and the location of user file contents on storage targets.
- It offers scale-out services, allowing for one or many metadata services in a BeeGFS file system.
- Each metadata service is responsible for its exclusive fraction of the global namespace. Having more metadata servers improves the overall system performance.
- Usually, a metadata target is an ext4 file system based on a RAID1 or RAID10 of flash drives. 512GB of usable metadata capacity is typically sufficient for about 150 million user files.
- Using faster CPU cores will result in low metadata access latency.

Storage service

- The storage service (sometimes also referred to as the “object storage service”) is the main service to store striped user file contents, also known as data chunk files.
- The Storage Service scale-out design allows for one or multiple storage services per BeeGFS file system instance in turn allowing for more capacity and performance to the file system.
- A storage service instance has one or multiple storage targets.
- The storage target can generally be any directory on a local file system (usually xfs); a storage target typically is a hardware RAID-6 (typically composed of 8+2 or 10+2) or zfs RAIDz2 volume, of either internal or externally attached drives.

Client service

- BeeGFS comes with a client that registers natively with the virtual file system interface of the Linux kernel for maximum performance.
- The Client kernel module must be compiled to match the used kernel.
- When the Client module is loaded using the Client service, it will mount the file systems defined in beegfsmounts.conf.
- The Client kernel module uses an additional user space helper daemon for DNS lookups and to write the log file.

xiRAID Overview

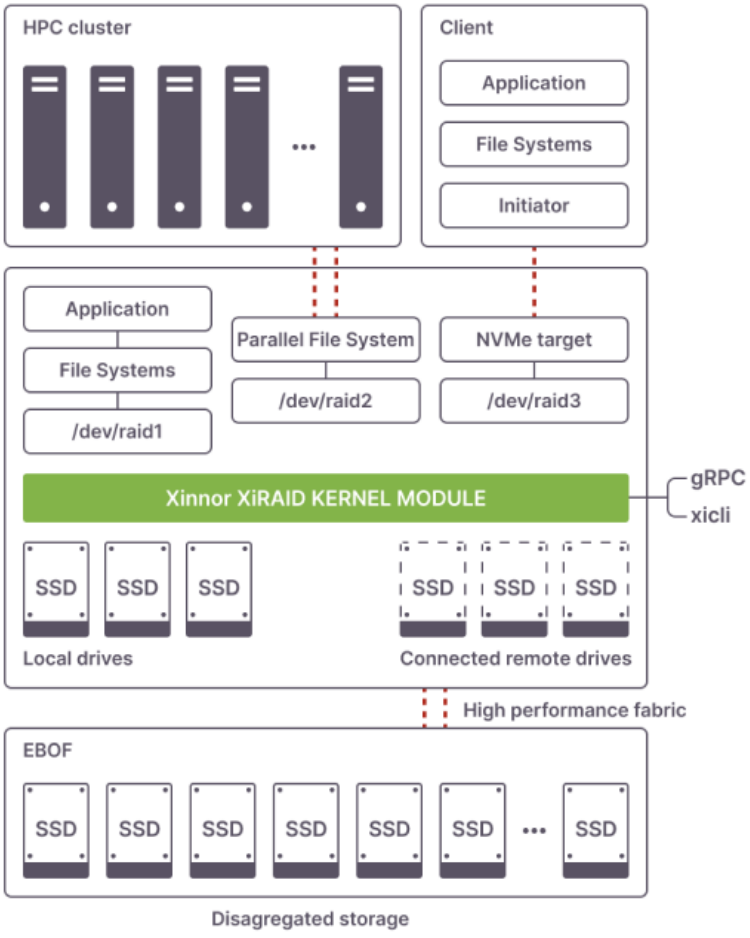
xiRAID ensures fast and effective access to data by allowing for the creation of high-performance RAID from NVMe and SAS/SATA SSD’s. Designed for the most demanding enterprise-grade tasks, xiRAID is easy to maintain and suited for operating in large server infrastructures. For more details, refer to @Xinnor.

xiRAID is a software RAID presented by Linux kernel module and management utility (CLI).

- Adjusted for the most popular Linux® distribution (Ubuntu, RHEL, Oracle® Linux, Rocky Linux, Alma Linux).
- Works with local and remote drives.
- Provides RAID as a standard Linux block device.
- POSIX API support.

I/O handling parallelization and lockless data path in xiRAID allows for the removal of array internal barriers and deliver unprecedented performance. xiRAID is also able to sustain high performance levels and low latency (< 0.5ms) even in mixed workloads. To protect data, xiRAID delivers a wide range of RAID level support: RAID 1/0/5/6/7.3/50/60/70.

Moreover, drive failure causes a low performance loss, which helps business applications run smoothly. That comes from an innovative approach to erasure coding calculations.



High Speed of the Checksums Calculations

The core innovation of the Xinnor product is the unique software RAID that calculates array parity faster than any other alternative in the storage industry. The RAID engine reads and writes parity blocks with the record speed (about 25GBps for 1 CPU core) and therefore maintains high array performance even during a degraded RAID state.

Reduced RAID Rebuild Time

Rebuild (or reconstruction) of the RAID after a drive failure is a potentially fraught and dangerous time frame for storage administrators. Reconstructing data to a new drive usually consumes a significant percentage of the total array performance. There is also the increased risk of data loss during the rebuild phase as the acceptable number of drives available for failure is reduced. With fast checksums calculation, Xinnor software arrays require significantly less time to perform rebuild operations compared to existing solutions in the global storage market.

Advantages of the Xinnor Software RAID

Due to its fast coding and decoding ability, xiRAID provides the stable performance levels needed for smooth and uninterrupted business operations. Fast RAID rebuild protects storage from extensive system downtime and mitigates the impact on workflows.

This is crucial for data-intensive systems and high-density storage infrastructures where even a single drive failure can cause checksum recalculations for a vast amount of data.

BeeGFS Solution Design on OpenFlex Data24

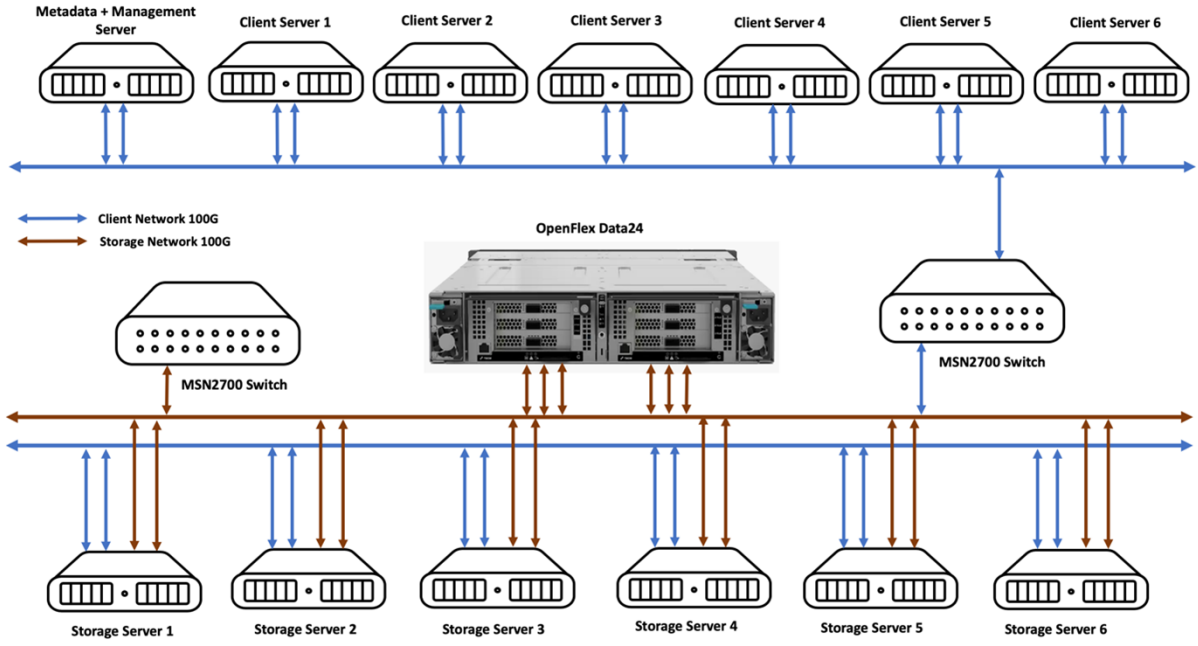
Western Digital has partnered with BeeGFS to provide a verified, highly scalable shared network file system with striped file contents, high-performance computing and mirroring (HA) for on-premises and cloud systems.

The following sections of this paper provide an overview of the storage solution.

BeeGFS on OpenFlex Data24 Deployment Topology

OpenFlex Data24 and BeeGFS have created a unique reference architecture with industry-standard servers and ethernet network switches. It leverages the OCI approach in the form of disaggregated data storage using NVMe-over-Fabrics (NVMe-oF).

OpenFlex Data24 uses NVMe-oF technology with the NVMe-RoCEv2 protocol for all network transmissions. This end-to-end solution can be specifically designed to accelerate large-scale production while delivering the compute and storage performance needed. The figure below illustrates a verified deployment topology with OpenFlex Data24.



Hardware Requirements

BeeGFS Storage Server Details	
Storage Product	OpenFlex Data24 with 24 x 3.84 TB Ultrastar DC SN840 SSDs
Storage Interface	Six QSFP28 (100Gbps) fabric ports
Host OS	Red Hat Enterprise Linux® release 8.4(Ootpa)
Kernel	4.18.0-305.el8.x86_64
Host NIC	2 x CX5 - MCX516A-CCAT (100Gbps)
NIC Firmware Version	16.32.1010
CX5 OFED package version	5.5-1.0.3
CPU	AMD® EPYC® 7352 24-Core Processor
CPU core details	Dual socket server with 24 core CPU each. 96 logical cores in total with HT enabled
Memory	128GiB
Number of volumes on each storage server	4

BeeGFS Management / Metadata Server Details	
Host OS	Red Hat Enterprise Linux release 8.4 (Ootpa)
Kernel	4.18.0-305.el8.x86_64
Host NIC	1 x CX5 - MCX516A-CCAT (100Gbps)
CX5 OFED package version	5.5-1.0.3
CPU	AMD EPYC 7352 24-Core Processor
CPU Core Details	Dual socket server with 24 core CPU each. 96 logical cores in total with HT enabled
NIC Firmware Version	16.32.1010
Memory	128GB
NVMe Disk	2 x 1.92TB for Management service 4 x 3.84TB for Metadata service

BeeGFS Client Server Details	
Host OS	Red Hat Enterprise Linux release 8.3(Ootpa)
Kernel	4.18.0-240.el8.x86_64
Host NIC	1 x CX5 - MCX516A-CCAT (100Gbps)
CX5 OFED package version	5.4-2.4.1
CPU	Intel® Xeon® Gold 5318Y CPU @ 2.10GHz
CPU Core Details	Dual socket server with 24 core CPU each. 96 logical cores in total with HT enabled
NIC Firmware Version	16.31.2006
Memory	128GiB

BeeGFS deployment with OpenFlex Overview

- Configure the OpenFlex Data24 system.
- Configure the storage servers, metadata servers, and client servers' prerequisites.
- Install BeeGFS packages on all servers (management, metadata, storage, and client).
- Set up BeeGFS services on all servers (management, metadata, storage, and client).
- Apply tuning parameters to improve performance.
- Pre-requisites on all servers
  - Configure NTP on all servers and sync the servers.
  - Disable Selinux on all systems. Edit the below file and set the value to disabled.

```
$ vi /etc/selinux/config
$ SELINUX=disabled
```

  - Reboot the server after disabling Selinux.
  - Install Mellanox firmware. Check that firmware details installed as per the versions specified in the hardware configuration.
  - Add the below configuration if same subnet used during IP configuration for both the ports of the CX5 card to avoid ARP flux issues.

```
$ ifconfig ens3f0 192.168.5.18/24 mtu 9216 up
$ ifconfig ens3f1 192.168.5.28/24 mtu 9216 up

$ sysctl -w net.ipv4.conf.ens3f0.arp_ignore=1
$ sysctl -w net.ipv4.conf.ens3f0.arp_filter=0
$ sysctl -w net.ipv4.conf.ens3f0.arp_announce=2
$ sysctl -w net.ipv4.conf.ens3f0.rp_filter=0

$ sysctl -w net.ipv4.conf.ens3f1.arp_ignore=1
$ sysctl -w net.ipv4.conf.ens3f1.arp_filter=0
$ sysctl -w net.ipv4.conf.ens3f1.arp_announce=2
$ sysctl -w net.ipv4.conf.ens3f1.rp_filter=0

$ ip neigh flush dev ens3f0
$ ip neigh flush dev ens3f1

$ echo 201 ens3f0 >> /etc/iproute2/rt_tables
$ echo 202 ens3f1 >> /etc/iproute2/rt_tables

$ ip route add 192.168.5.0/24 dev ens3f0 proto kernel scope link src 192.168.5.18 table ens3f0
$ ip route add 192.168.5.0/24 dev ens3f1 proto kernel scope link src 192.168.5.28 table ens3f1
$ ip rule add from 192.168.5.18 table ens3f0
$ ip rule add from 192.168.5.28 table ens3f1
$ ip route flush cache
```

Note: On the storage nodes add the details of the cluster interface only.

Configure OpenFlex Data24

Refer to Openflex-Data24-NVMe-oF-Storage-Platform for more details. Follow the steps mentioned in OpenFlex Data24 User Guide to set up the OpenFlex system. For more details regarding lossless configuration on OpenFlex contact @WD Support.

- Set up lossless configuration on OpenFlex server. Lossless settings also need to be configured on the switch and servers. For details regarding lossless configuration contact @WD Support.
- Connect nvme device to the storage server.

```
$  nvme connect -t rdma -s 4420 -a 192.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
$  nvme connect -t rdma -s 4420 -a 172.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
```

- Get the size of nvme device.

```
$ nvme id-ctrl /dev/nvme1 | grep -i tnvmcap
  tnvmcap   : 3840755982336
```

- Create nvme namespace on the nvme device.

```
$ nvme create-ns -s 3840755982336 -c 3840755982336 -f 0x2 -nmic 1 /dev/nvme1
```

- Create 1 namespace each on all the 24drives using commands mentioned above. In total create 24 namespaces out of 24 drives on OpenFlex Data24.
- Map all the Data24 namespaces using the above steps to appropriate storage servers using the steps below:
- To map the nvme device using multipath get both controller ids.

```
$ nvme id-ctrl /dev/nvme1 | grep cntl
  cntlid    : 0x1
$ nvme id-ctrl /dev/nvme2 | grep cntl
  cntlid    : 0x8001
```

- Connect the nvme namespace using both controller ids to set multipath.

```
$ nvme attach-ns /dev/nvme1 -n 1 -c 1
$ nvme attach-ns /dev/nvme2 -n 1 -c 0x8001
```

Configure Storage Server

Lossless Configuration

- Set up lossless configuration on the host servers by running the below commands on all the BeeGFS storage servers:

```
modprobe nvme_core multipath=yes
modprobe nvme
modprobe nvme-rdma
modprobe nvme-core
modprobe nvme-fabrics
mst start
mlnx_qos -i ens6f0 --trust dscp
mlnx_qos -i ens6f1 --trust dscp
mlnx_qos -i ens6f0 --pfc 0,0,0,1,0,0,0,0
mlnx_qos -i ens6f1 --pfc 0,0,0,1,0,0,0,0
mlnx_qos -i ens6f0 --prio2buffer 0,0,0,1,0,0,0,0
mlnx_qos -i ens6f1 --prio2buffer 0,0,0,1,0,0,0,0
mlnx_qos -i ens6f0 --tsa ets,ets,ets,ets,ets,ets,strict,ets --tcbw 14,15,14,15,14,14,0,14
mlnx_qos -i ens6f1 --tsa ets,ets,ets,ets,ets,ets,strict,ets --tcbw 14,15,14,15,14,14,0,14
cma_roce_mode -d mlx5_3 -m 2
cma_roce_mode -d mlx5_2 -m 2
cma_roce_tos -d mlx5_3 -t 96
cma_roce_tos -d mlx5_2 -t 96
cma_roce_tos -d mlx5_3
cma_roce_tos -d mlx5_2
mlxlink -d /dev/mst/mt4119_pciconf0.1 -s 100G --link_mode_force
mlxlink -d /dev/mst/mt4119_pciconf0 -s 100G --link_mode_force
```

- Configure lossless settings on the switch and on all BeeGFS storage servers where volumes from OpenFlex Data24 platform will be mapped. For more details regarding lossless configuration on OpenFlex platform and switches, visit @WD Support and follow the steps mentioned in the lossless configuration guide for your switch brand.
- Set up the IP addresses. Set the MTU value to 4500 for all storage ports and the MTU value to 9216 for other ports on the host as well as on the switch. Verify all the ports are up and configured properly.

Set up Native NVMe Multipath on all Storage Servers

- Check that native NVMe multipathing is enabled in the kernel.

```
$ [root@node1 ~]#   cat /sys/module/nvme_core/parameters/multipath
  N
```

- If the above commands show N then add the below content to “/etc/modprobe.d/nvme_core.conf” file to enable multipath.

```
$ options nvme_core multipath=1 io_timeout=1
```

- Connect the namespaces from OpenFlex Data24 and configure the multipath IO policy.

```
$ [root@node1 ~]#  nvme connect -t rdma -s 4420 -a 192.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
$ [root@node1 ~]#  nvme connect -t rdma -s 4420 -a 172.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
```

```
[root@node1 ~]# nvme list
Node             SN                   Model                                    Namespace Usage
Format           FW Rev
---------------- -------------------- ---------------------------------------- --------- ------------------
/dev/nvme0n1     S61ENA0R203384       Dell Ent NVMe AGN MU U.2 1.6TB           1           1.60  TB /  1.60
TB    512   B +  0 B   2.0.2
/dev/nvme1n1     A05D3C74             WUS4BA138DSP3X1                          1           3.84  TB /
3.84  TB      4 KiB +  0 B   R2210004
```

- Add the below contents to "/etc/udev/rules.d/71-nvmf-iopolicy-data24.rules" file to configure round-robin IO policy for NVMe namespaces mapped from OpenFlex Data24.

```
ACTION=="add", SUBSYSTEM=="nvme-subsystem", ATTR{model}=="WUS4BA138DSP3X1", ATTR{iopolicy}="round-robin"
```

Disconnect all the mapped namespaces/volumes.

```
$ root@node1 ~]#  nvme disconnect-all
```

Unload all the NVMe modules using below commands.

```
[root@node1 ~]#  sudo modprobe -r nvme-rdma
[root@node1 ~]#  sudo modprobe -r nvme-fabrics
[root@node1 ~]#  sudo modprobe -r nvme
[root@node1 ~]#  sudo modprobe -r nvme-core
```

Load the NVMe modules again.

```
[root@node1 ~]#  sudo modprobe nvme
[root@node1 ~]#  sudo modprobe nvme-rdma
```

Reconnect all the namespaces/volumes again.

```
[root@node1 ~]#   nvme connect -t rdma -s 4420 -a 192.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
[root@node1 ~]#  nvme connect -t rdma -s 4420 -a 172.168.100.11 -i 16 -q host\
-n nqn.1992-05.com.wdc.openflex-data24:nvme.1
```

- Verify that native NVMe multipath is enabled and configured IO-policy is also applied.

```
[root@node1 ~]#   cat /sys/module/nvme_core/parameters/multipath
Y
[root@node1 ~]#   cat /sys/class/nvme-subsystem/nvme-subsys*/iopolicy
round-robin
```

- Check that the mapped NVMe namespaces shows multipath details.

```
[root@node1 ~]# nvme list-subsys
nvme-subsys0 - NQN=nqn.1994-11.com.samsung:nvme:PM1733:2.5-inch:S61ENA0R203384
\
+- nvme0 pcie 0000:62:00.0 live
nvme-subsys1 - NQN=nqn.1992-05.com.wdc.openflex-data24:nvme.1
\
+- nvme1 rdma traddr=192.168.100.11 trsvcid=4420 live
```

Set up Management / Metadata Server

- Set up the IP. Set up MTU value to 9216 for the ports on host and switch.
- Install NVMe-CLI. Check the newly added disks for metadata are getting listed.

Set up Xinnor xiRAID Software on all Storage/Metadata Servers

Obtain a valid license and download the xiRAID utility on the servers. The xiRAID distribution consists of four software packages for Linux OS.

- xiraid – the main functionality which is the driver for RAID arrays.
- xiraid-util – a user management utility for RAIDs and licenses.

The package type depends on your Linux distribution. To get the license and software distribution packages refer @Xinnor.

- After downloading the zip file, to install xiRAID 4.0, unpack the archive packages with the distribution to a folder and go to the corresponding system OS folder.

```
$ unzip 'XIRAID dkms..zip'
```

```
$ cd "XIRAID dkms/Install/ CentOS-RHEL 8/xiraid"
```

- To install xiRAID 4.0, run the installation script installer.sh, which is in the archive with program packages.

```
$ ./installer.sh
```

- While running the script it will ask you to accept the terms of the license agreement. You will see the text of the agreement on your screen. It will ask you to confirm the installation of DKMS. Check the presence of the required packages with the kernel header files. Install the required dependencies from the repositories. For more details refer to the Xinnor RAID Administration Guide.
- If the installation fails, then check and install all the required dependencies. After installing all required dependencies, rerun the ./ installer.sh script. Once the installation is complete, it will ask you to start the xiRAID services.
- To ensure that the packages were installed successfully, run the below command, and check the status.

```
$ xiraid show
```

- Once you have your license file, copy it to the server and apply the license key by running the command before proceeding with xiRAID installation and configuration.

```
$ xiraid license --update /root/license.txt
```

- Repeat the above steps on all the storage servers and metadata servers to install xiRAID software.

Set up Client Server

- Set up the IPs. Set up MTU value to 9216 for the ports on host and switch.
- Repeat the above steps in all client servers.

Note: If the configuration meets all the requirements, you are ready to set up BeeGFS.

BeeGFS Deployment Details

This section provides detailed configuration steps and settings for all the BeeGFS components using multimode configuration. Find more details regarding the configuration refer to the BeeGFS Configuration Steps. For single instance configuration refer to the BeeGFS Manual Install Walk Through.

BeeGFS Configuration

The BeeGFS test configuration includes the following components:

- Management server
- Metadata server
- Storage server
- Client server

The complete deployment is tested on OpenFlex Data24 with xiRAID and BeeGFS. Based on BeeGFS recommendations, the metadata server is configured to use ext4 file system, and the storage servers are configured to use XFS file systems on top of xiRAID volumes.

In BeeGFS, multi-mode is a way to run multiple instances of one type of service on one server. This is useful if more than one BeeGFS file system/storage target instance or more than one metadata target instance is configured on a single physical server. Note, when using multi-mode within a single file system, BeeGFS will treat them as individual machines.

For multi-mode, you will need to create a separate configuration file for the other daemon instance, using different network ports, a different storage directory, a different log file, and so on. If the second daemon instance on a machine should become part of the same file system instance (i.e., it registers at the same management daemon as the first daemon instance on this machine), then you would also need to set a different NodeID manually for the second daemon instance.

BeeGFS Management Configuration Details

Setting up the BeeGFS management service.

- Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo\
https://www.beegfs.io/release/beegfs_7.3.0/dists/beegfs-rhel8.repo
```

- Install BeeGFS management packages.

```
$ yum install beegfs-mgmtd
```

- A xiRAID1 volume is used to host the management data.

Create the RAID1 volume and wait until the initialization is over to proceed further.

```
$ xiraid create {-n|--name} <raid_name> {-l|--level} <raid_level> {-d|--drives} (block_devices)
$ xiraid create -n era1 -l 1 -d /dev/nvme1n1 /dev/nvme2n1 -ss 64 -bs 4096
```

```
[xinnor@mgmt:~$ xicli raid show
RAIDs
| name | static           | state       | devices                  | info                 |
| era1 | size: 982 GiB    | online      | 0 /dev/nvme1n1 online    | memory_usage_mb  : - |
|      | level: 1         | initialized | 1 /dev/nvme2n1 online    |                      |
|      | strip_size: 64   |             |                          |                      |
|      | block_size: 4096 |             |                          |                      |
|      | sparepool: -     |             |                          |                      |
|      | active: True     |             |                          |                      |
|      | config: True     |             |                          |                      |
```

- Create a partition on the disk/volume on the management server. You can configure ext3, ext4 or XFS partition as per your preference.

```
$ [root@mgmt ~]# mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype /dev/xiraid_era1
mke2fs 1.45.6 (20-Mar-2020)
Creating filesystem with 468794448 4k blocks and 937590784 inodes
Filesystem UUID: bda770a5-bf9a-4116-a9f7-038239d58139
Superblock backups stored on blocks:
16384, 49152, 81920, 114688, 147456, 409600, 442368, 802816, 1327104,
2048000, 3981312, 5619712, 10240000, 11943936, 35831808, 39337984,
51200000, 107495424, 256000000, 275365888, 322486272

Allocating group tables: done
Writing inode tables: done
Creating journal (102400 blocks): done
Writing superblocks and filesystem accounting information: done
```

- Create directory which you want to use as the mount point for the management.

```
$ mkdir -p  /mnt/beegfsmgmt/
```

- Mount the volume and check the mount details.

```
$ root@mgmt ~]# mount -onoatime,nodiratime,nobarrier /dev/xiraid_era1 /mnt/beegfsmgmt/
$ [root@mgmt ~]# mount
/dev/xiraid_era1 on /mnt/beegfsmgmt type ext4 (rw,noatime,nodiratime,nobarrier,stripe=16)
/dev/xiraid_era2 on /data/beegfs/beegfs_meta1 type ext4 (rw,noatime,nodiratime,nobarrier,stripe=16)
/dev/xiraid_era3 on /data/beegfs/beegfs_meta2 type ext4 (rw,noatime,nodiratime,nobarrier,stripe=16)
```

- Set up the management location to the newly created directory.

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p <management_data_location>
$ [root@mgmt ~]# /opt/beegfs/sbin/beegfs-setup-mgmtd -p /mnt/beegfsmgmt
Preparing storage directory: /mnt/beegfsmgmt
 * Creating format.conf file...
Updating config file: /etc/beegfs/beegfs-mgmtd.conf
 * Setting storage directory in config file...
 * Disabling usage of uninitialized storage directory in config file...
All done.
```

- Create the “connInterfacesFile.conf” file in the config directory.

Note: This file will hold the network interfaces to be used with BeeGFS services. Provide the network interface details of the management node in connInterfacesFile.conf file.

```
$ [root@mgmt]# cat /etc/beegfs/connInterfacesFile.conf
               ens31f0
```

- Edit the BeeGFS configuration file and set the location for connInterfacesFile. You specify which network interface can be used to contact the management service. Create a text file under /etc/beegfs, with the interface name to be used, listed one per line. In the following example, ens31f0 is used. The absolute path of the file must be specified in the configuration file /etc/beegfs/beegfs-mgmt.conf by using the connInterfacesFile parameter.

```
$ [root@mgmt1 ~]# cat /etc/beegfs/connInterfacesFile.conf
                  ens31f0
```

- Configure the location of the management data.

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p <mgmt_data_location> -c <mgmt_config_location>\
-S <beegfs_mgmt_string_nodeID>
```

Example:

```
$ /opt/beegfs/sbin/beegfs-setup-mgmtd -p /mnt/beegfsmgmt/ -c /etc/beegfs/beegfs-mgmtd.conf -S mgmt-server
```

- Start the management service and enable it to start at boot.

```
$ systemctl start beegfs-mgmtd.service
$ systemctl enable beegfs-mgmtd.service
$ systemctl status beegfs-mgmtd.service
```

BeeGFS MetaData Configuration Details

Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo\
https://www.beegfs.io/release/beegfs_7.3.0/dists/beegfs-rhel8.repo
```

Install metadata packages.

```
$ yum install beegfs-meta libbeegfs-ib
```

- xiRAID1 volumes are used to host the metadata instances. To set up the xiRAID1 volumes follow below steps.

Create the RAID1 volumes and wait until the initialization is over to proceed further.

```
$ xiraid create {-n|--name} <raid_name> {-l|--level} <raid_level> {-d|--drives} (block_devices)
$ xiraid create -n era2 -l 1 -d /dev/nvme3n1 /dev/nvme4n1 -ss 64 -bs 4096
$ xiraid create -n era3 -l 1 -d /dev/nvme5n1 /dev/nvme6n1 -ss 64 -bs 4096
```

```
[xinnor@mgmt:~$ xicli raid show
RAIDs
```

name	static	state	devices	info
era1	size: 982 GiB level: 1 strip_size: 64 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme1n1 online 1 /dev/nvme2n1 online	memory_usage_mb : -
era2	size: 982 GiB level: 1 strip_size: 64 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme3n1 online 1 /dev/nvme6n1 online	memory_usage_mb : -
era3	size: 982 GiB level: 1 strip_size: 64 block_size: 4096 sparepool: - active: True config: True	online initialized	0 /dev/nvme7n1 online 1 /dev/nvme8n1 online	memory_usage_mb : -

- Create directories which you want to use as mount points for the metadata. In the current deployment we have configured two metadata instances.

```
$ mkdir -p /data/beegfs/beegfs_meta1 /data/beegfs/beegfs_meta2
```

- Format the volumes where you want to store the metadata.

```
$ mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype <mapped volume>
$ [root@mgmt ~]# mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype /dev/xiraid_era2
mke2fs 1.45.6 (20-Mar-2020)
Creating filesystem with 937635408 4k blocks and 1875279872 inodes
Filesystem UUID: 76fab1e7-8d6c-45f5-9197-147e99a9a681
Superblock backups stored on blocks:
16384, 49152, 81920, 114688, 147456, 409600, 442368, 802816, 1327104,
2048000, 3981312, 5619712, 10240000, 11943936, 35831808, 39337984,
51200000, 107495424, 256000000, 275365888, 322486272
```

```
Allocating group tables: done
Writing inode tables: done
Creating journal (102400 blocks): done
Writing superblocks and filesystem accounting information: done

$ [root@mgmt ~]# mkfs.ext4 -i 2048 -I 512 -J size=400 -Odir_index,filetype /dev/xiraid_era3
mke2fs 1.45.6 (20-Mar-2020)
Creating filesystem with 937635408 4k blocks and 1875279872 inodes
Filesystem UUID: 9aa8141b-9aa3-4d3c-aa0b-40106a3e62ff
Superblock backups stored on blocks:
16384, 49152, 81920, 114688, 147456, 409600, 442368, 802816, 1327104,
2048000, 3981312, 5619712, 10240000, 11943936, 35831808, 39337984,
51200000, 107495424, 256000000, 275365888, 322486272

Allocating group tables: done
Writing inode tables: done
Creating journal (102400 blocks): done
Writing superblocks and filesystem accounting information: done
```

- Mount each mapped volume.

```
$ mount -onoatime,nodiratime,nobarrier <mapped volume> <mount point>.
[root@mgmt ~]# mount -onoatime,nodiratime,nobarrier /dev/xiraid_era2 /data/beegfs/beegfs_meta1
[root@mgmt ~]# mount -onoatime,nodiratime,nobarrier /dev/xiraid_era3 /data/beegfs/beegfs_meta2
```

- Add each mount point to /etc/fstab to automatically mount when the server boots up.
- As you are configuring two metadata instances here on the same server for multimode configuration, you need to run the following steps.

```
$ mkdir /etc/beegfs/inst1.d
$ mkdir /etc/beegfs/inst2.d
$ cp /etc/beegfs/beegfs-meta.conf /etc/beegfs/inst1.d/
$ cp /etc/beegfs/beegfs-meta.conf /etc/beegfs/inst2.d/
$ vi /etc/beegfs/inst1.d/beegfs-meta.conf # Adapt “connMetaPort*”, “logStdFile”, etc.
$ vi /etc/beegfs/inst2.d/beegfs-meta.conf # Adapt “connMetaPort*”, “logStdFile”, etc.
```

- You specify which network interface or interfaces other services can use to contact the metadata services. Create a text file under /etc/beegfs, with the interface names to be used, listed one per line.

In the following example, ens31f0, and ens31f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/inst1.d/beegfs-meta.conf and /etc/beegfs/inst2.d/beegfs-meta.conf by using the connInterfacesFile parameter.

```
$ [root@mgmt ~]# cat /etc/beegfs/connInterfacesFile1.conf
                ens31f0
                ens31f1
$ [root@mgmt ~]# cat /etc/beegfs/inst1.d/beegfs-meta.conf | grep connInterfacesFile
connInterfacesFile          = /etc/beegfs/connInterfacesFile1.conf

$ [root@mgmt ~]# cat /etc/beegfs/inst2.d/beegfs-meta.conf | grep connInterfacesFile
```

```
connInterfacesFile           = /etc/beegfs/connInterfacesFile1.conf
```

- Configure metadata service by running the following steps.

```
$ /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst1.d/beegfs-meta.conf -p        /data/beegfs/beegfs_
meta1 -s 11 -S meta1-inst1 -m node01
```

To add a second metadata target on the same machine change the instance path, id, and name and run as the below command.

```
$ [root@mgmt ~]# /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst1.d/beegfs-meta.conf\
-p /data/beegfs/beegfs_meta1 -s 11 -S meta1-inst1 -m 192.168.5.11
Preparing storage directory: /data/beegfs/beegfs_meta1
* Creating format.conf file...
* Creating server numeric ID file: /data/beegfs/beegfs_meta1/nodeNumID
* Creating server string ID file: /data/beegfs/beegfs_meta1/nodeID
Updating config file: /etc/beegfs/inst1.d/beegfs-meta.conf
* Setting management host: 192.168.5.11
* Setting storage directory in config file...
* Disabling usage of uninitialized storage directory in config file...
* Fetching the underlying device...
Underlying device detected: /dev/xiraid_era2
Fetching UUID of the file system on that device...
Found UUID 76fab1e7-8d6c-45f5-9197-147e99a9a681
Writing UUID to config file...
* Setting usage of extended attributes to: true
All done.

$[root@mgmt ~]# /opt/beegfs/sbin/beegfs-setup-meta -c /etc/beegfs/inst2.d/beegfs-meta.conf\
-p /data/beegfs/beegfs_meta2 -s 12 -S meta1-inst2 -m 192.168.5.11
```

- Start the metadata service and enable it to start at boot.

```
$ [root@mgmt ~]# systemctl start beegfs-meta@inst1
$ [root@mgmt ~]# systemctl enable beegfs-meta@inst1
$ [root@mgmt ~]# systemctl status beegfs-meta@inst1
$ [root@mgmt ~]# systemctl start beegfs-meta@inst2
$ [root@mgmt ~]# systemctl enable beegfs-meta@inst2
$ [root@mgmt ~]# systemctl status beegfs-meta@inst2
```

- Repeat above steps on the node which you want to configure as a metadata node

Storage Server Configuration Details

- Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo\
https://www.beegfs.io/release/beegfs_7.3.0/dists/beegfs-rhel8.repo
```

- Install storage packages.

```
$ yum install beegfs-storage libbeegfs-ib
# libbeegfs-ib is only required for RDMA
```

- xiRAID RAID5 volumes are used to host the storage instance. To set up the xiRAID RAID5 volumes follow below steps.

```
$ xiraid create {-n|--name} <raid_name> {-l|--level} <raid_level> {-d|--drives} (block_devices)
```

Note: Once the RAID5 volumes are created wait until the initialization is over to proceed further.

- Set merge enabled (me), Schedule Enabled (se) and merge_wait_usecs (mw) and merge_max_usecs (mm)values as 9000 for the RAID5 volumes.

```
$ xiraid  modify -n era1 -me 1 -mw 9000 -mm 9000 -se 1
```

```
root@node1:/home/xinnor# xicli raid show -e
RAIDs
```

name	static	state	devices	health	wear	serials	params	info
era1	size: 2946 GiB level: 5 strip_size: 64 block_size: 4096 sparepool: - active: True config: True	online initing	0 /dev/nvme0n1 online 1 /dev/nvme1n1 online 2 /dev/nvme3n1 online 3 /dev/nvme2n1 online	100% 100% 100% 100%	2% 5% 5% 4%	A068F544_1 A068F6C0_1 A068F6A2_1 A068F4AF_1	init_prio : 100 init_depth : 128 recon_prio : 100 recon_depth : 64 memory_limit_mb : 0 merge_read_enabled : 0 merge_write_enabled: 1 resync_enabled : 1 sched_enabled : 1 request_limit : 0 restripe_prio : 100 merge_wait_usecs : 9000 merge_max_usecs : 9000	init_progress : 39 memory_usage_mb : -

- Create directories which you want to use as mount points for the storage servers. We are using one instance here for the storage server. Based on your requirements, you can configure multiple storage instances and point them to different volumes and mount points and configure multimode BeeGFS storage instances.

```
$ mkdir -p /mnt/target211/beegfs_storage
```

- Format each mapped volume.

```
$ mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k <mapped volume>
```

Note: “su” is the segment size of the volume.

```
$ [root@node1 ~]# mkfs.xfs -d su=128k,sw=8 -l version=2,su=128k /dev/xi _ era1
mkfs.xfs: Specified data stripe unit 256 is not the same as the volume stripe unit 128
mkfs.xfs: Specified data stripe width 2048 is not the same as the volume stripe width 384
meta-data=/dev/xiraid_era1          isize=512    agcount=32, agsize=87903328 blks=sectsz=4096 attr=2,
projid32bit=1
=crc=1        finobt=1, sparse=1, rmapbt=0
=reflink=1
data     =bsize=4096   blocks=2812906224, imaxpct=5
=sunit=32     swidth=256 blks
naming   =version 2              bsize=4096   ascii-ci=0, ftype=1
log=internal log           bsize=4096   blocks=521728, version=2
=sectsz=4096  sunit=32 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

- Mount each mapped volume.

```
$ root@node1 ~]# mount -o
```

- Add each mount point to /etc/fstab to automatically mount when the server boots up.
- In the current deployment we are configuring only one BeeGFS storage instance. But based on requirements you can configure multiple storage instances using multimode configuration.

```
$ mkdir /etc/beegfs/inst1.d
```

```
$ cp /etc/beegfs/beegfs-storage.conf /etc/beegfs/inst1.d/
$ vi /etc/beegfs/inst1.d/beegfs-storage.conf # Adapt “connStoragePort*”, “logStdFile”, etc.
```

Edit the network port details, storage directory path, and log file path details in the above files if you want to configure multiple storage instances.

- Specify which network interface or interfaces other services can use to contact the storage services. Create a text file under /etc/beegfs, with the interface names to be used, listed one per line. In the following example, ens2f0 and ens2f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/inst1.d/beegfs-storage.conf by using the connInterfacesFile parameter.

```
$ [root@node1 ~]# cat /etc/beegfs/connInterfacesFile.conf
ens2f0
ens2f1
$ [root@ storage1 ~]# cat /etc/beegfs/inst1.d/beegfs-storage.conf | grep connInterfacesFile
connInterfacesFile = /etc/beegfs/connInterfacesFile.conf
```

- Configure the storage service by running the following steps

```
$ [root@node1 ~]# /opt/beegfs/sbin/beegfs-setup-storage -c /etc/beegfs/inst1.d/beegfs-storage.conf\
-p /mnt/target211/beegfs_storage -s 21 -S stor1-inst1 -i 211 -m 192.168.5.11
Preparing storage target directory: /mnt/target211/beegfs_storage
* Creating format.conf file...
* Creating chunks directory...
* Creating buddymir directory...
* Creating target numeric ID file: /mnt/target211/beegfs_storage/targetNumID
* Creating server numeric ID file: /mnt/target211/beegfs_storage/nodeNumID
```

```
* Creating server string ID file: /mnt/target211/beegfs_storage/nodeID
Updating config file: /etc/beegfs/inst1.d/beegfs-storage.conf
* Setting management host: 192.168.5.11
* Appending to target directory list in config file...
* Disabling usage of uninitialized storage targets in config file...
* Fetching the underlying device...
Underlying device detected: /dev/xiraid_era1
Fetching UUID of the file system on that device...
Found UUID de2ae772-b758-473d-9325-d06fdc05ecaa
Appending UUID to config file...
All done.
```

- Start the storage service and enable it to start at boot.

```
$ [root@node1 ~]# systemctl start beegfs-storage@inst1
$ [root@node1 ~]# systemctl enable beegfs-storage@inst1
$ [root@node1 ~]# systemctl status beegfs-storage@inst1
beegfs-storage@inst1.service - BeeGFS Storage Server (multimode)
Loaded: loaded (/usr/lib/systemd/system/beegfs-storage@.service; disabled; vendor preset: disabled)
Active: active (running) since Wed 2022-05-25 01:12:09 CDT; 1s ago
Docs: http://www.beegfs.com/content/documentation/
Main PID: 3704811 (beegfs-storage/)
Tasks: 88 (limit: 822020)
Memory: 58.0M
CGroup: /system.slice/system-beegfs\x2dstorage.slice/beegfs-storage@inst1.service
           3704811 /opt/beegfs/sbin/beegfs-storage cfgFile=/etc/beegfs/inst1.d/beegfs-storage.conf
runDaemonized=false

May 25 01:12:09 node1 systemd[1]: Started BeeGFS Storage Server (multimode).
```

- Repeat the above steps on all the storage nodes.

Client Server Configuration Details

- Add BeeGFS repository.

```
$ yum install https://dl.fedoraproject.org/pub/epel/epel-release-latest-8.noarch.rpm
$ wget -O /etc/yum.repos.d/beegfs_rhel8.repo\
https://www.beegfs.io/release/beegfs_7.3.0/dists/beegfs-rhel8.repo
```

- Install client packages.

```
$ yum install beegfs-client beegfs-helperd beegfs-utils
```

- Client kernel module auto build:

To enable support for remote direct memory access (RDMA) based on the OFED ibverbs API, the corresponding BeeGFS client kernel module build parameters need to be added. If you installed separate OFED kernel modules, add the OFED_INCLUDE_PATH:

```
$ ssh clientnode
$ vi /etc/beegfs/beegfs-client-autobuild.conf
$ buildArgs=-j8  OFED_INCLUDE_PATH=/usr/src/ofa_kernel/default/include
```

- Afterwards, force a rebuild of the client kernel module.

```
$ /etc/init.d/beegfs-client rebuild
```

- The client mount directory is defined in a separate configuration file. This file will be used by the beegfs-client service start-up script. By default, BeeGFS will be mounted to /mnt/beegfs. Thus, you need to perform this step only if you want to mount the file system to a different location.

```
$ ssh root@clientnode
$ vi /etc/beegfs/beegfs-mounts.conf
```

The first entry defines the mount directory. The second entry refers to the corresponding config file for this mountpoint.

Else create a mount point as below where BeeGFS data will get mounted.

```
$ mkdir -p /mnt/beegfs
```

- You need to specify which network interface or interfaces other services can use to contact the storage services. Create a text file under /etc/beegfs, with the interface names to be used listed one per line. In the following example, ens31f0, and ens31f1 are used. The absolute path of the file must be specified in the configuration file /etc/beegfs/ and /etc/beegfs/beegfs-client.conf by using the connInterfacesFile parameter.

```
$ [root@client1 ~]# cat /etc/beegfs/connInterfacesFile.conf
ens31f0
ens31f1
```

- The client needs to know where the management service is running.

```
$ /opt/beegfs/sbin/beegfs-setup-client -m node01
$ /opt/beegfs/sbin/beegfs-setup-client -m 192.168.5.11
```

- Start the client services.

```
$ [root@client1 ~]# systemctl start beegfs-helperd.service
$ [root@client1 ~]# systemctl enable beegfs-helperd.service
```

```
$ [root@client1 ~]# systemctl status beegfs-helperd.service
$ [root@ client1 ~]# systemctl start beegfs-client.service
$ [root@ client1 ~]# systemctl enable beegfs-client.service
$ [root@ client1 ~]# systemctl status beegfs-client.service
```

- Repeat the above steps in all the client nodes.

BeeGFS Configuration Status Details

- Check BeeGFS system configuration details.

```
$ root@client1 ~]# beegfs-df
METADATA SERVERS:
TargetID   Cap. Pool        Total         Free    %      ITotal       IFree    %
========   =========        =====         ====    =      ======       =====    =
           11      normal    2681.7GiB    2681.4GiB 100%     1875.3M     1875.2M 100%
           12      normal    2681.7GiB    2681.4GiB 100%     1875.3M     1875.2M 100%

STORAGE TARGETS:
TargetID   Cap. Pool        Total         Free    %      ITotal       IFree    %
========   =========        =====         ====    =      ======       =====    =
           211      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
           311      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
           411      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
           511      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
           611      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
           711      normal   10728.4GiB   10653.6GiB  99%     1125.2M     1125.2M 100%
[root@client1 ~]# beegfs-net

mgmt_nodes
==============
mgmt [ID: 1]
           Connections: TCP: 1 (192.168.5.11:8008);

meta_nodes
==============
meta1-inst1 [ID: 11]
           Connections: RDMA: 1 (192.168.5.11:8005);
meta1-inst2 [ID: 12]
    Connections: <none>

storage_nodes
==============
stor1-inst1 [ID: 21]
```

```
Connections: RDMA: 1 (192.168.5.13:8003);
stor2-inst1 [ID: 31]
Connections: RDMA: 1 (192.168.5.14:8003);
stor3-inst1 [ID: 41]
Connections: RDMA: 1 (192.168.5.15:8003);
stor4-inst1 [ID: 51]
Connections: RDMA: 1 (192.168.5.16:8003);
stor5-inst1 [ID: 61]
Connections: RDMA: 1 (192.168.5.17:8003);
stor6-inst1 [ID: 71]
Connections: RDMA: 1 (192.168.5.18:8003);

$ [root@client1 ~]# beegfs-ctl --listnodes --nodetype=client --details
13D3D-628DD202-dhcp-10-206-132-145 [ID: 1]
Ports: UDP: 8004; TCP: 0
Interfaces: ens31f0(RDMA) ens31f0(TCP) ens31f1(TCP) ens31f1(RDMA)
1B4C8-628DD20E-dhcp-10-206-133-214 [ID: 2]
Ports: UDP: 8004; TCP: 0
Interfaces: ens5f0(RDMA) ens5f0(TCP) ens5f1(TCP) ens5f1(RDMA)
1B38C-628DD219-dhcp-10-206-133-134 [ID: 3]
Ports: UDP: 8004; TCP: 0
Interfaces: ens5f0(RDMA) ens5f0(TCP) ens5f1(TCP) ens5f1(RDMA)
1B43C-628DD20F-dhcp-10-206-132-146 [ID: 4]
Ports: UDP: 8004; TCP: 0
Interfaces: ens5f0(RDMA) ens5f0(TCP) ens5f1(TCP) ens5f1(RDMA)
13280-628DD210-dhcp-10-206-132-184 [ID: 5]
Ports: UDP: 8004; TCP: 0
Interfaces: ens5f0(RDMA) ens5f0(TCP) ens5f1(TCP) ens5f1(RDMA)
13345-628DD211-client1 [ID: 6]
Ports: UDP: 8004; TCP: 0
Interfaces: ens5f0(RDMA) ens5f0(TCP) ens5f1(TCP) ens5f1(RDMA)

Number of nodes: 6

$ [root@client1 ~]# beegfs-ctl --listnodes --nodetype=storage --details
stor1-inst1 [ID: 21]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens2f0(RDMA) ens2f0(TCP) ens2f1(RDMA) ens2f1(TCP)
stor2-inst1 [ID: 31]
Ports: UDP: 8003; TCP: 8003
Interfaces: enp161s0f0(RDMA) enp161s0f0(TCP) enp161s0f1(RDMA) enp161s0f1(TCP)
stor3-inst1 [ID: 41]
Ports: UDP: 8003; TCP: 8003
Interfaces: enp161s0f0(RDMA) enp161s0f0(TCP) enp161s0f1(RDMA) enp161s0f1(TCP)
stor4-inst1 [ID: 51]
```

```
Ports: UDP: 8003; TCP: 8003
Interfaces: enp161s0f0(RDMA) enp161s0f0(TCP) enp161s0f1(RDMA) enp161s0f1(TCP)
stor5-inst1 [ID: 61]
Ports: UDP: 8003; TCP: 8003
Interfaces: enp161s0f0(RDMA) enp161s0f0(TCP) enp161s0f1(RDMA) enp161s0f1(TCP)
stor6-inst1 [ID: 71]
Ports: UDP: 8003; TCP: 8003
Interfaces: ens3f0(RDMA) ens3f0(TCP) ens3f1(RDMA) ens3f1(TCP)

Number of nodes: 6
$ [root@client1 ~]# beegfs-ctl --listnodes --nodetype=meta --details
meta1-inst1 [ID: 11]
Ports: UDP: 8005; TCP: 8005
Interfaces: ens31f0(RDMA) ens31f0(TCP) ens31f1(RDMA) ens31f1(TCP)
meta1-inst2 [ID: 12]
Ports: UDP: 8006; TCP: 8006
Interfaces: enp161s0f1(RDMA) enp161s0f1(TCP) enp161s0f0(RDMA) enp161s0f0(TCP)

Number of nodes: 2
Root: 11
```

- At runtime, you can check whether your RDMA devices have been discovered by using beegfs-ctl by listing all registered services and their configured network interfaces in order of preference:

```
$ beegfs-ctl --listnodes --nodetype=storage --details
$ beegfs-ctl --listnodes --nodetype=meta --details
$ beegfs-ctl --listnodes --nodetype=client –details
```

- To check the BeeGFS log for information and any error messages refer below files.

```
$ less /var/log/beegfs-mgmtd.log
$ less /var/log/beegfs-meta.log
$ less /var/log/beegfs-storage.log
$ less /var/log/beegfs-client.log
```

Tuning Parameters to Improve Performance

Xinnor Tuning Parameters

- In the system BIOS enable below values. (The options availability depends on the BIOS version).
- Set System C-states value to disabled
- Set PCIe Maximum Payload to 256b
- Set PCIe Maximum Read Request to 4096b

Under Linux kernel boot options, you can add the below parameters.

Note: The below parameters are optional. Adding these options will increase the system performance by sacrificing security. If you decide to add these options, you need to regenerate grub.cfg with grub2-mkconfig.

Add the following values in the Linux kernel options to the GRUB_CMDLINE_LINUX at /etc/default/grub:

```
$ "noibrs noibpb nopti nospectre_v2 nospectre_v1 l1tf=off nospec_store_bypass_disable no_stf_barrier\
mds=off tsx=on tsx_async_abort=off mitigations=off"
```

- Run below commands on the system.

```
$ modprobe -r nvme && modprobe nvme poll_queues=36
$ echo 0 | tee /sys/block/nvme*n1/queue/io_poll_delay
$ echo 0 | tee /sys/block/nvme*n1/queue/iostats
```

Before initiating any tests on the xiRAID volumes, follow the below recommendations:

- Wait until a RAID initialization is complete before starting a performance test.
- Use “throughput-performance” tuned-adm profile (if tuned is installed in your system).

BeeGFS Storage Server Tuning

Note: Refer to @StorageServerTuning for more details.

- After mapping nvme devices run the below tuning parameters.

```
echo 5 > /proc/sys/vm/dirty_background_ratio
echo 20 > /proc/sys/vm/dirty_ratio
echo 50 > /proc/sys/vm/vfs_cache_pressure
echo 262144 > /proc/sys/vm/min_free_kbytes
echo 1 > /proc/sys/vm/zone_reclaim_mode

echo always > /sys/kernel/mm/transparent_hugepage/enabled
echo always > /sys/kernel/mm/transparent_hugepage/defrag

devices=(nvme1n1 nvme3n1 nvme5n1 nvme7n1)
for dev in "${devices[@]}"
do
echo deadline > /sys/block/${dev}/queue/scheduler
echo 2048 > /sys/block/${dev}/queue/nr_requests
echo 4096 > /sys/block/${dev}/queue/read_ahead_kb
echo 64 > /sys/block/${dev}/queue/max_sectors_kb
done
```

- On the server in /etc/beegfs/inst1.d/beegfs-storage.conf

```
$ tuneNumWorkers=80
$ tuneBindToNumaZone=0
```

Note: Change the tuneNumWorkers value as per the number CPU cores available in your system. You can change the value to 4 or 8 times the number of available CPU cores and tune it to achieve better system performance.

BeeGFS Metadata Server Tuning

Note: Refer to @Metadata_tuning for more details.

- To keep the configuration permanently, you could add the corresponding commands to /etc/rc.local, as seen in the example below. Use /etc/sysctl.conf or create udev rules to reapply them automatically when the machine boots.

```
#!/bin/bash
echo 5 > /proc/sys/vm/dirty_background_ratio
echo 20 > /proc/sys/vm/dirty_ratio
echo 50 > /proc/sys/vm/vfs_cache_pressure

echo 262144 > /proc/sys/vm/min_free_kbytes
echo 1 > /proc/sys/vm/zone_reclaim_mode
```

```
echo always > /sys/kernel/mm/transparent_hugepage/enabled
echo always > /sys/kernel/mm/transparent_hugepage/defrag
devices=(nvme1n1 nvme2n1 nvme3n1 nvme4n1 nvme5n1 nvme6n1)
for dev in "${devices[@]}"
do
echo deadline > /sys/block/${dev}/queue/scheduler
echo 128 > /sys/block/${dev}/queue/nr_requests
echo 128 > /sys/block/${dev}/queue/read_ahead_kb
echo 256 > /sys/block/${dev}/queue/max_sectors_kb
done
```

System BIOS Settings

```
$ echo performance | tee /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor >/dev/null
```

Each metadata server establishes multiple connections to each of the other servers to enable more parallelism on the network level by having multiple requests in flight to the same server. The tuning option tuneNumWorkers in /etc/beegfs/beegfs-meta.conf can be used to configure the number of simultaneous connections.

Note: Change the tuneNumWorkers value as per the CPU core available in your system.

BeeGFS Client Tuning

Note: Refer to @Client_Tuning for more details.

- On the client in /etc/beegfs/beegfs-client.conf.

```
$ connRDMABufSize=8192
$ connRDMABufNum=70
$ connMaxInternodeNum=64
```

Note: Change the connMaxInternodeNum value as per the CPU core available in your system.

- Increase the chunksize from standard 512k to 1M.

```
$ beegfs-ctl --setpattern --chunksize=1M --numtargets=4 /mnt/beegfs
New chunksize: 1048576
New number of storage targets: 4
Path:
Mount: /mnt/beegfs
```

Note: Based on the chunk size and targets you want to stripe the data and change the values.

- On the client in /etc/beegfs/beegfs-client.conf increase the cache buffer size on all client nodes.

```
$ tuneFileCacheBufSize = 2097152
```

- Restart all the BeeGFS services so that all the tuning parameter changes will get reflected across all servers.

Communication Retries:
BeeGFS clients support a time-limited retry mode for communications, which will keep on trying to complete the I/O operation in case a server is temporarily unreachable. The connCommRetrySecs option in /etc/beegfs/beegfs-client.conf sets the time limit for communication retries.

Client Multi-RAIL Support

The default behaviour of the BeeGFS client is to use a single RDMA NIC for communications with beegfs-meta and beegfs-storage. When the client has multiple RDMA NICs, it is advantageous to configure multi-rail support.

BeeGFS 7.3.0 introduces explicit multi-rail support in the BeeGFS client via the connRDMAInterfacesFile configuration in /etc/beegfs/beegfs-client.conf.

Explicit multi-rail support provides the following features:

- Specification of which client RDMA NICs to use for BeeGFS RDMA traffic.
- Client makes use of multiple RDMA NICs in a single IPoIB subnet.
- Dynamic load-balancing between RDMA NICs according to connection count.
- Support for selecting an RDMA NIC according to GPUDirect Storage Support NVFS device priority.

Multi-rail support will not work correctly if the client uses RDMA NICs configured in separate IPoIB subnets. Every RDMA NIC configured for client use must have IPoIB connectivity to every BeeGFS service in the cluster. Multi-rail support does not depend upon “connInterfacesFile”. “connRDMAInterfacesFile” specifies the path to a file containing the names of devices the BeeGFS client should use for outbound RDMA connections. The file format is one IPoIB device name listed per line.

```
/etc/beegfs/beegfs-client.conf:

connRDMAInterfacesFile = /etc/beefgs/client-rdma.conf

$ cat /etc/beegfs/client-rdma.conf:

ens2f0

ens2f1
```

When configuring a node’s multiple IPoIB devices in the same IPv4 subnet, it may be required to configure the IP routing tables and rules for multi-homed support. The primary indicator of the need for this configuration is when BeeGFS client cannot RDMA connect to all of the BeeGFS services. This is not the same task as enabling IP-forwarding between NICs; it is a routing configuration to segregate traffic between NICs on the same IPv4 subnet. Improve Your Multi-Home Servers With Policy Routing discusses multi-homed IPv4 configuration in detail.

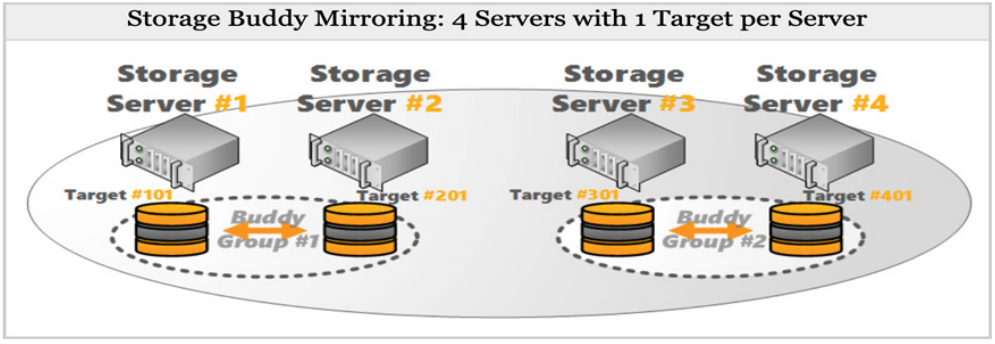
The following sysctl parameters are useful for multi-home:

```
$ net.ipv4.conf.all.rp_filter = 1

$ net.ipv4.conf.all.arp_filter = 1

$ net.ipv4.conf.all.arp_announce = 2

$ net.ipv4.conf.all.arp_ignore = 2

$ net.ipv4.conf.default.rp_filter = 1

$ net.ipv4.conf.default.arp_filter = 1

$ net.ipv4.conf.default.arp_announce = 2

$ net.ipv4.conf.default.arp_ignore = 2
```

These changes may need to be performed on the client and/or server nodes.

BeeGFS Buddy Mirroring Configuration

BeeGFS supports mirroring of data across buddy groups. BeeGFS provides support for metadata and file content mirroring. Mirroring capabilities are integrated into the normal BeeGFS services, so that no separate services or third-party tools are needed. Both types of mirroring (metadata mirroring and file contents mirroring) can be used independently of each other. Refer to @BuddyGroups and @MetaDataMirror for more details.



Automatic creation of buddy groups can be done with beegfs-ctl, separately for metadata and for storage servers:

```
$ beegfs-ctl --addmirrorgroup --automatic --nodetype=meta
$ beegfs-ctl --addmirrorgroup --automatic --nodetype=storage
```

Note: See the built-in help of beegfs-ctl for more information on available parameters.

Defining Storage Buddy Mirror Groups Manually

Manual definition of mirror buddy groups can be useful if you want to set custom group IDs or if you want to make sure that the buddies are in different failure domains. Manual definition of mirror buddy groups is done with the beegfs-ctl tool. By using the following command, you can create a buddy group with the ID 100, consisting of targets 1 and 2.

```
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=1 --secondary=2 --groupid=100
$ beegfs-ctl --addmirrorgroup --nodetype=storage --primary=211 --secondary=311 --groupid=100
```

To List-Defined Buddy Mirrored Groups

```
$ beegfs-ctl --listmirrorgroups --nodetype=storage
```

To Display Target States

```
$ beegfs-ctl --listmirrorgroups --nodetype=storage
```

Based on your requirement you can configure different types of buddy groups.

To Define Stripe Pattern for Mirroring

BeeGFS supports folder level mirroring with different chunk size for striping. You can set the number of targets and chunk size based on your requirement for mirroring. Refer to BeeGFS BuddyMirroring for more information.

Use the below commands to set the chunksize to stipe the data across specific targets.

```
$ beegfs-ctl --setpattern --buddymirror --help
$ [root@client1 ~]# beegfs-ctl --setpattern --chunksize=1m  --pattern=buddymirror --numtargets=2\
/mnt/beegfs/bdm
```

Validate the buddy mirroring configuration for the folder.

```
$ [root@client1 ~]#  beegfs-ctl --getentryinfo /mnt/beegfs/bdm
```

BeeGFS Metadata Mirroring

To active metadata mirroring, use the beegfs-ctl tool.

```
$ beegfs-ctl –mirrormd
```

Running this command will enable metadata mirroring for the root directory of the BeeGFS, as well as all the files contained in it. Note that existing directories except for the root directory will not be mirrored automatically.

Please see the help of beegfs-ctl for more information on available parameters.

```
$ beegfs-ctl --mirrormd –help
```

Notes: When applying the beegfs-ctl --mirrormd command, no clients may be mounted. This can be achieved by stopping the beegfs-client service beforehand, and starting it again after beegfs-ctl --mirrormd was successfully performed. In addition, please restart the beegfs-meta service on all nodes afterwards.

List metadata mirroring groups.

```
$ beegfs-ctl --listmirrorgroups --nodetype=meta
```

BeeGFS Metadata Sync

If a secondary target or server is considered to be out-of-sync, you can initiate resync using the below commands.

```
$ beegfs-ctl --startresync --help
$ beegfs-ctl --resyncstats --help
```

The following command could be used to stop the automatic resynchronization and start a full resynchronization instead.

```
$ beegfs-ctl --startresync --nodetype=storage --targetid=X --timestamp=0 --restart
```

BeeGFS Node Timeout Settings

BeeGFS support a time-limited retry mode for communications. Set the timeout value to avoid delay in case of failure when the storage/metadata servers are temporarily unreachable; the buddy mirrored secondary target will become the primary server and handle all the requests. You need to set the timeout value in the management and client servers as well to reinitiate connectivity in case of access failure.

The default setting is 180 seconds on the server. Edit the sysTargetOfflineTimeoutSecs value and decrease the timeout value to 30/60 seconds to avoid delay in case of node failure. Edit the timeout value for management, metadata, and client nodes as well to avoid access failure/delay in case of connectivity issues.

Storage Node file location

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/inst1.d/beegfs-storage.conf
sysTargetOfflineTimeoutSecs = 180
```

Management Node file location

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/beegfs-mgmt.conf
sysTargetOfflineTimeoutSecs = 180
```

MetadataNode file location

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/inst1.d/beegfs-meta.conf
sysTargetOfflineTimeoutSecs = 60
```

Client Node file location.

```
$sysTargetOfflineTimeoutSecs /etc/beegfs/beegfs-client.conf
sysTargetOfflineTimeoutSecs = 900
```

Performance Test with IOR

IOR is a benchmark tool to measure performance of a single or multiple clients with one or more processes per client. IOR is based on MPI for distributed execution. It can be used to measure streaming throughput or small random IO performance (IOPS). Refer to @IOR for more details. Follow the instruction below to set up IOR.

Configuring IOR

If configure is missing from the top-level directory, you probably retrieved this code directly from the repository. Run ./bootstrap to generate the configure script.

1. Run ./configure. For a full list of configuration options, use ./configure --help.
2. Run “make”, then run “make install.” The installation prefix can be changed via ./configure --prefix=....
3. Export the following paths:

```
$ export PATH=$PATH:/usr/mpi/gcc/openmpi-4.0.3rc4/bin/
$ export OMPI_ALLOW_RUN_AS_ROOT=1
$ export OMPI_ALLOW_RUN_AS_ROOT_CONFIRM=1
```

Testing

IOPS benchmark example

```
$ mpirun -hostfile /tmp/nodefile --map-by node -np ${NUM_PROCS} /usr/bin/IOR -w -i5 -t4k -b \
${BLOCK_SIZE} -F -z -g -o /mnt/beegfs/test.ior
```

mpirun parameters

`-hostfile $PATH` (file with the hostnames of the clients/servers to benchmark)

`-np $N` (number of processes)

IOR parameters

`-w` (write benchmark)

`-r` (read benchmark)

`-i $N` (repetitions)

`-t $N` (transfer size, for dd it is the block size)

`-b $N` (block size, amount of data for a process)

`-g` (use barriers between open, write/read, and close)

`-e` (perform fsync upon POSIX write close, make sure reads are only started are all writes are done.)

`-o $PATH` (path to file for the test)

`-F` (one file per process)

`-z` (random access to the file)

IOR test results on on xiRAID volumes

```
$ [root@client1 ~]# mpirun -hostfile /tmp/nodefile6 --map-by node -np 288 /usr/local/bin/ior -i3 -t1M\
-b 35G -F -g -e -o /mnt/beegfs/1m/am7.ior
IOR-3.3.0: MPI Coordinated Test of Parallel I/O
Began               : Fri May 27 03:03:22 2022
Command line        : /usr/local/bin/ior -i3 -t1M -b 35G -F -g -e -o /mnt/beegfs/1m/am7.ior
Machine             : Linux dhcp-10-206-132-145
TestID              : 0
StartTime           : Fri May 27 03:03:22 2022
Path                : /mnt/beegfs/1m
FS                  : 62.9 TiB   Used FS: 0.7%   Inodes: 0.0 Mi   Used Inodes: -nan%
```

```
      Options:
      api                 : POSIX
      apiVersion          :
      test filename       : /mnt/beegfs/1m/am7.ior
      access              : file-per-process
      type                : independent
      segments            : 1
      ordering in a file  : sequential
      ordering inter file : no tasks offsets
      nodes               : 6
      tasks               : 288
      clients per node    : 48
      repetitions         : 3
      xfersize            : 1 MiB
      blocksize           : 35 GiB
      aggregate filesize  : 9.84 TiB
Results:
access    bw(MiB/s)  IOPS       Latency(s)  block(KiB) xfer(KiB)  open(s)    wr/rd(s)   close(s)   total(s)
iter
------    ---------  ----       ----------  ---------- ---------  --------   --------   --------   --------
----
write     21030      21032      0.013507    36700160   1024.00    0.028338   490.77     0.018395   490.82     0
read      38415      38421      0.006881    36700160   1024.00    0.022984   268.66     0.022035   268.70     0
remove    -          -          -           -          -          -          -          -          3.53       0
write     20975      20977      0.011989    36700160   1024.00    0.028218   492.05     0.024765   492.11     1
read      40157      40164      0.006240    36700160   1024.00    0.028594   256.99     0.027569   257.04     1
remove    -          -          -           -          -          -          -          -          3.48       1
write     22344      22347      0.012887    36700160   1024.00    0.024206   461.90     0.031274   461.96     2
read      38764      38771      0.007039    36700160   1024.00    0.033800   266.23     0.033480   266.28     2
remove    -          -          -           -          -          -          -          -          3.25       2
Max Write: 22343.93 MiB/sec (23429.31 MB/sec)
Max Read:  40157.03 MiB/sec (42107.70 MB/sec)


Summary of all tests:
Operation   Max(MiB)   Min(MiB)  Mean(MiB)     StdDev   Max(OPs)   Min(OPs)  Mean(OPs)     StdDev    Mean(s)
Stonewall(s) Stonewall(MiB) Test# #Tasks tPN reps fPP reord reordoff reordrand seed segcnt   blksiz    xsize
aggs(MiB)   API RefNum
write       22343.93   20974.94   21449.62     632.77   22343.93   20974.94   21449.62     632.77  481.62749
NA            NA     0    288  48    3   1     0        1         0    0      1 37580963840  1048576 10321920.0
POSIX      0
read        40157.03   38414.64   39111.77     752.72   40157.03   38414.64   39111.77     752.72  264.00498
NA            NA     0    288  48    3   1     0        1         0    0      1 37580963840  1048576 10321920.0
POSIX      0
Finished            : Fri May 27 03:40:50 2022
```

IOR Performance Results

The maximum performance achieved with IOR without buddy mirroring on xiRAID RAID5 volumes.

BeeGFS performance on xiRAID RAID5 volumes (without Buddy Mirror)

IOR Test Results	GBps
Read	42 GBps
Write	24 GBps

The maximum performance achieved with IOR with buddy mirroring on NON-RAID volumes:

BeeGFS performance on Non-RAID volumes with Buddy Mirror

IOR Test Results	GBps
Read	45 GBps
Write	7.7 GBps

The maximum performance achieved with IOR on without buddy mirroring on NON-RAID volumes. This configuration lacks redundancy and presents a risk of data-loss:

BeeGFS performance on Non-RAID volumes (without Buddy Mirror)

IOR Test Results	GBps
Read	45 GBps
Write	30 GBps

Document Purpose

This document provides instructions on how to set up a BeeGFS configuration quickly and easily with OpenFlex Data24 using Xinnor xiRAID RAID volumes. The measured performance numbers are specific to the current configuration. This reference architecture provides high throughput levels with the IOR tests using a set of files that begin to approach the limits of the storage controllers and drives. The tests give insight into performance with different file counts and sizes. The test results are intended to show the upper range of performance expectations.

Conclusion

BeeGFS on the Western Digital's OpenFlex Data24 Storage with xiRAID addresses the need of IT/HPC with a well-designed solution that is easy to manage and fully supported. The solution includes the added benefit of the open composable infrastructure environment and enables disaggregated compute on the Nvme-oF storage platform. BeeGFS on the Western Digital's OpenFlex Data24 systems shows excellent benchmark results for storage data streaming with both RAID and Non-RAID configuration. This highlights the superiority of Nvme-oF devices over traditional technology for high-end services with different workloads.

In this document we have explained the deployment details and the performance achieved with BeeGFS using xiRAID on OpenFlex Data24. The test results clearly shows that BeeGFS, Xinnor xiRAID, and OpenFlex Data24 has significant advantages of performance and ease of deployment. Additionally, we have shown that implementing xiRAID, users will be able to achieve significant performance with data protection and high availability, and gain extra capacity for user data. This new software technology is well suited to accelerate storage hardware systems to new levels of high performance.

By leveraging the solution offered by Western Digital's enterprises and service providers can

- Minimize the cost of the installation
- Implement high-performance HA file system solution
- Accelerate time to market for new application services

Using this solution based on the latest NVMe-oF and CDI technologies, organizations can quickly deploy a proven, self-service, composable infrastructure solution, helping customers move to a more flexible, variable cost model.

Appendix

Contributors

Name	Company	Title
Puspanjali Panda	Western Digital	Principal Engineer Test Engineer
Saravanakumar Pandian	Western Digital	Principal Engineer Test Engineer
Pavan Gururaj	Western Digital	Senior Manager Test Engineer
Niall MacLeod	Western Digital	Director Applications Engineer

Document Feedback

For feedback, questions, and suggestions for improvements to this document send an email to the Data Center Systems (DCS) Technical Marketing Engineering (TME) team distribution list at pdl-dcs-tm@wdc.com.

Version History

Version	Revision Date	Notes
01	February 2023	Initial Release
02	May 2023	General textual updates throughout the document

References

- https://www.westerndigital.com/en-in/products/data-center-platforms/openflex-data24-nvme-of-platform#vvc-capacity-368_TB
- https://documents.westerndigital.com/content/dam/doc-library/en_us/assets/public/western-digital/product/platforms/openflex/product-brief-data24-nvme-of-storage-platform-gov.pdf
- https://doc.beegfs.io/latest/index.html
- https://www.beegfs.io/release/beegfs_7.3.0/dists/
- https://doc.beegfs.io/latest/advanced_topics/multimode.html
- https://doc.beegfs.io/latest/advanced_topics/rdma_support.html
- https://doc.beegfs.io/latest/advanced_topics/storage_tuning.html
- https://doc.beegfs.io/latest/advanced_topics/metadata_tuning.html
- https://doc.beegfs.io/latest/advanced_topics/client_tuning.html
- https://doc.beegfs.io/latest/advanced_topics/rdma_support.html#rdmasupport
- https://www.usenix.org/system/files/login/articles/login_summer16_10_anderson.pdf
- https://ior.readthedocs.io/en/latest/intro.html
- https://github.com/hpc/ior
- https://xinnor.io/files/Xinnor_xiRAID_Administrators_Guide.pdf

5601 Great Oaks Parkway
San Jose, CA 95119, USA
www.westerndigital.com