Distributed Asynchronous Object Storage (DAOS)

Installation, Configuration and Administration Guide

Extreme Storage Architecture & Development (ESAD) Division

March 2019

NOTICES

Generated under Argonne Contract number**: 8F-30005**

Acknowledgment: "This material is based upon work supported by the U.S. Department of Energy and Argonne National Laboratory and its Leadership Computing Facility, including under Contract DE-AC02-06CH11357 and Award Number 8F-30005. This work was generated with financial support from the U.S. Government through said Contract and Award Number(s), and as such the U.S. Government retains a paid-up, nonexclusive, irrevocable, world-wide license to reproduce, prepare derivative works, distribute copies to the public, and display publicly, by or on behalf of the Government, this work in whole or in part, or otherwise use the work for Federal purposes."

Company Name: Intel Federal LLC

Company Address: 4100 Monument Corner Drive, Suite 540 Fairfax, VA 22030

Document Revision History

Revision Number	Date	Comments
0.5	March 2019	Initial version.
.65	March 2019	Document updates and feedback incorporated
1.0	May 2019	Document notices revised, first release of community document

Contents

Tables

Figures

1 Introduction

The Distributed Asynchronous Object Storage (DAOS) is an open-source object store designed from the ground up for massively distributed Non Volatile Memory (NVM). DAOS takes advantage of next-generation NVM technology, like Storage Class Memory (SCM) and NVM express (NVMe), while presenting a key-value storage interface on top of commodity hardware that provides features, such as, transactional non-blocking I/O, advanced data protection with self-healing, end-to-end data integrity, fine-grained data control, and elastic storage, to optimize performance and cost.

This initial administration guide version is associated with DAOS v0.4.

1.1 Terms used in this Document

The following terms and abbreviations are used in this document.

Term	Definition
ACLs	Access Control Lists
CaRT	Collective and RPC Transport (CaRT) library. A software library built on top of the Mercury Function Shipping library to support distributed communication functionality.
CGO	Go tools that enable creation of Go packages that call C code
CN	Compute Node
CPU	Central Processing Unit
COTS	Commercial off-the-shelf
Daemon	A process offering system-level resources.
DCPM	Intel Optane DC Persistent Memory
DPDK	Data Plane Development Kit
dRPC	DAOS Remote Procedure Call
BIO	Blob I/O
gRPC	gRPC[1] Remote Procedure Calls
GURT	A common library of Gurt Useful Routines and Types provided with CaRT.
HLD	High-Level Design
I/O	Input/Output
ISA-L	Intel® Intelligent Storage Acceleration Library
libfabric	A user-space library that exports the Open Fabrics Interface
Mercury	A user-space RPC library that can use libfabrics as a transport
NVM	Non-Volatile Memory
NVMe	Non-Volatile Memory express
OFI	OpenFabrics Interfaces
OS	Operating System

[1] https://grpc.io/

Term	Definition
PMDK	Persistent Memory Development Kit
PMIx	Process Management Interface for Exascale
Raft	Raft is a consensus algorithm used to distribute state transitions among DAOS server nodes.
RDB	Replicated Database, containing pool metadata and maintained across DAOS servers using the Raft algorithm.
RPC	Remote Procedure Call.
SPDK	Storage Performance Development Kit
SWIM	Scalable Weakly-consistent Infection-style process group Membership protocol
UPI	Intel® Ultra Path Interconnect
UUID	Universal Unique Identifier
VOS	Versioned Object Store

1.2 Additional Documentation

Refer to the following documentation for architecture and description:

Document	Location
DAOS Internals	https://github.com/daos-stack/daos/blob/master/src/README.md
DAOS Storage Model	https://github.com/daos-stack/daos/blob/master/doc/storage_model.md
Community Roadmap	https://wiki.hpdd.intel.com/display/DC/Roadmap

2 DAOS Architecture

DAOS is an open source software-defined scale-out object store that provides high bandwidth and high IOPS storage containers to applications and enables next generation data-centric workflows combining simulation, data analytics and machine learning.

Unlike the traditional storage stacks that were primarily designed for rotating media, DAOS is architected from the ground up to exploit new NVM technologies and is extremely lightweight since it operates End-to-End (E2E) in user space with full OS bypass. DAOS offers a shift away from an I/O model designed for block-based and high-latency storage to one that inherently supports fine-grained data access and unlocks the performance of the next generation storage technologies.

Unlike traditional Burst Buffers, DAOS is a high-performant independent and fault-tolerant storage tier that does not rely on a third-party tier to manage metadata and data resilience.

2.1 DAOS Features

DAOS relies on OFI for low-latency communications and stores data on both storage-class memory and NVMe storage. DAOS presents a native key-array-value storage interface that offers a unified storage model over which domain-specific data models are ported, such as HDF5, MPI-IO and Apache Arrow. A POSIX I/O emulation layer implementing files and directories over the native DAOS API is also available.

DAOS I/O operations are logged and then inserted into a persistent index maintained in SCM. Each I/O is tagged with a particular timestamp called epoch and is associated with a particular version of the dataset. No read-modify-write operations are performed internally. Write operations are non-destructive and not sensitive to alignment. Upon read request, the DAOS service walks through the persistent index and creates a complex scatter-gather Remote Direct Memory Access (RDMA) descriptor to reconstruct the data at the requested version directly in the buffer provided by the application.

The SCM storage is memory-mapped directly into the address space of the DAOS service that manages the persistent index via direct load/store. Depending on the I/O characteristics, the DAOS service can decide to store the I/O in either SCM or NVMe storage. As represented in Figure 2-1, latency-sensitive I/Os, like application metadata and byte-granular data, will typically be stored in the former, whereas checkpoints and bulk data will be stored in the latter. This approach allows DAOS to deliver the raw NVMe bandwidth for bulk data by streaming the data to NVMe storage and maintaining internal metadata index in SCM. The Persistent Memory Development Kit (PMDK)[2] allows to manage transactional access to SCM and the Storage Performance Development Kit (SPDK)[3] enables user space I/O to NVMe devices.

[2] http://pmem.io/pmdk/
[3] http://www.spdk.io/

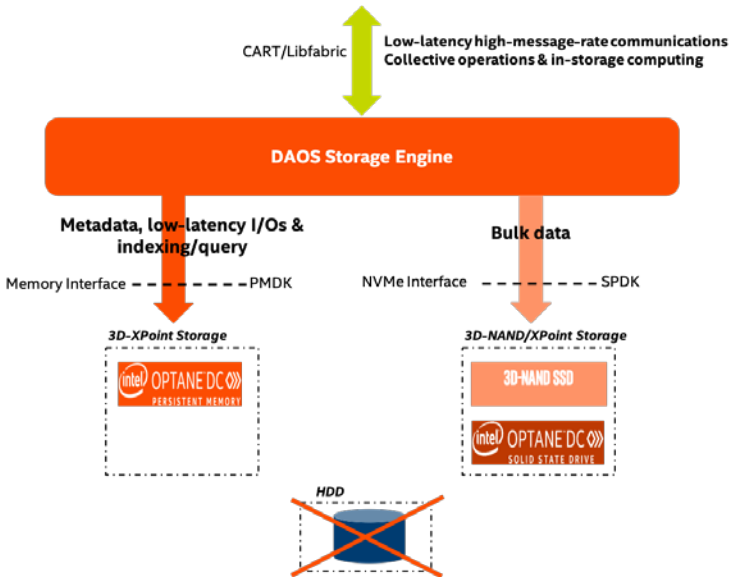


Figure 2-1. DAOS Storage

DAOS aims at delivering:

- High throughput and IOPS at arbitrary alignment and size
- Fine-grained I/O operations with true zero-copy I/O to SCM
- Support for massively distributed NVM storage via scalable collective communications across the storage servers
- Non-blocking data and metadata operations to allow I/O and computation to overlap
- Advanced data placement taking into account fault domains
- Software-managed redundancy supporting both replication and erasure code with online rebuild
- End-to-end data integrity
- Scalable distributed transactions with guaranteed data consistency and automated recovery
- Dataset snapshot
- Security framework to manage access control to storage pools
- Software-defined storage management to provision, configure, modify and monitor storage pools over COTS hardware
- Native support for Hierarchical Data Format (HDF)5, MPI-IO and POSIX namespace over the DAOS data model
- Tools for disaster recovery
- Seamless integration with the Lustre parallel filesystem
- Mover agent to migrate datasets among DAOS pools and from parallel filesystems to DAOS and vice versa

2.2 DAOS Components

A datacenter may have hundreds of thousands of compute nodes interconnected via a scalable high-performance fabric, where all, or a subset of the nodes called storage nodes, have direct access to NVM storage. A DAOS installation involves several components that can be either collocated or distributed.

2.2.1 DAOS Target, Server and System

The DAOS server is a multi-tenant daemon running on a Linux instance (i.e. natively on the physical node or in a VM or container) of each storage node and exporting through the network the locally-attached NVM storage. It listens to a management port, addressed by an IP address and a TCP port number, plus one or more fabric endpoints, addressed by network URIs. The DAOS server is configured through a YAML file and can be integrated with different daemon management or orchestration frameworks (e.g. a systemd script, a Kunernetes service or even via a parallel launcher like pdsh or srun).

A DAOS system is identified by a system name and consists of a set of DAOS servers connected to the same fabric. Membership of the DAOS servers is recorded into the system map that assigns a unique integer rank to each server. Two different systems comprise two disjoint sets of servers and do not coordinate with each other.

Inside a DAOS server, the storage is statically partitioned across multiple targets to optimize concurrency. To avoid contention, each target has its private storage, own pool of service threads and dedicated network context that can be directly addressed over the fabric independently of the other targets hosted on the same storage node. A target is typically associated with a single-ported SCM module and NVMe SSD attached to a single storage node. Moreover, a target does not implement any internal data protection mechanism against storage media failure. As a result, a target is a single point of failure. A dynamic state is associated with each target and is set to either up and running, or down and not available.

A target is the unit of performance. Hardware components associated with the target, such as the backend storage medium, the server, and the network, have limited capability and capacity.

The number of target exported by a DAOS server instance is configurable and depends on the underlying hardware (i.e. number of SCM modules, CPUs, NVMe SSDs ...). A target is the unit of fault.

2.2.2 Storage API, Application Interface and Tools

Applications, users and administrators can interact with a DAOS system through two different client APIs. The management API offers the ability to administrate a DAOS system. It is intended to be integrated with different vendor-specific storage management or open-source orchestration frameworks. A CLI tool is built over the DAOS management API. On the other hand, the DAOS library (i.e. libdaos) implements the DAOS storage model and is primarily targeted at application and I/O middleware developers who want to store datasets in a DAOS system. User utilities are also built over the API to allow users to manage datasets from a CLI.

Applications can access datasets stored in DAOS either directly through the native DAOS API or through an I/O middleware libraries (e.g. POSIX emulation, MPI-IO, HDF5) or frameworks (e.g. Spark, TensorFlow) already integrated with the native DAOS storage model.

2.2.3 Agent

The DAOS agent is a daemon residing on the client node that interacts with the DAOS library to authenticate the application process. It is a trusted entity that can sign the DAOS Client credentials using certificates. The agent can support different authentication frameworks and uses a Unix Domain Socket to communicate with the client library.

2.3 Storage Model

A DAOS pool is a storage reservation distributed across a collection of targets. The actual space allocated to the pool on each target is called a pool shard. The total space allocated to a pool is decided at creation time and can be expanded over time by resizing all the pool shards (within the limit of the storage capacity dedicated to each target) or by spanning more targets (i.e. adding more pool shards). A pool offers storage virtualization and is the unit of provisioning and isolation. DAOS pools cannot span across multiple systems.

A pool can host multiple transactional object store called DAOS containers. Each container is a private object address space, which can be modified transactional and independently of the other containers stored in the same pool. A container is the unit of snapshot and data management. DAOS objects belonging to a container can be distributed across any target of the pool for both performance and resilience and can be accessed through different APIs to efficiently represent structured, semi-structured and unstructured data.

Figure 2-2 illustrates the different DAOS abstractions.

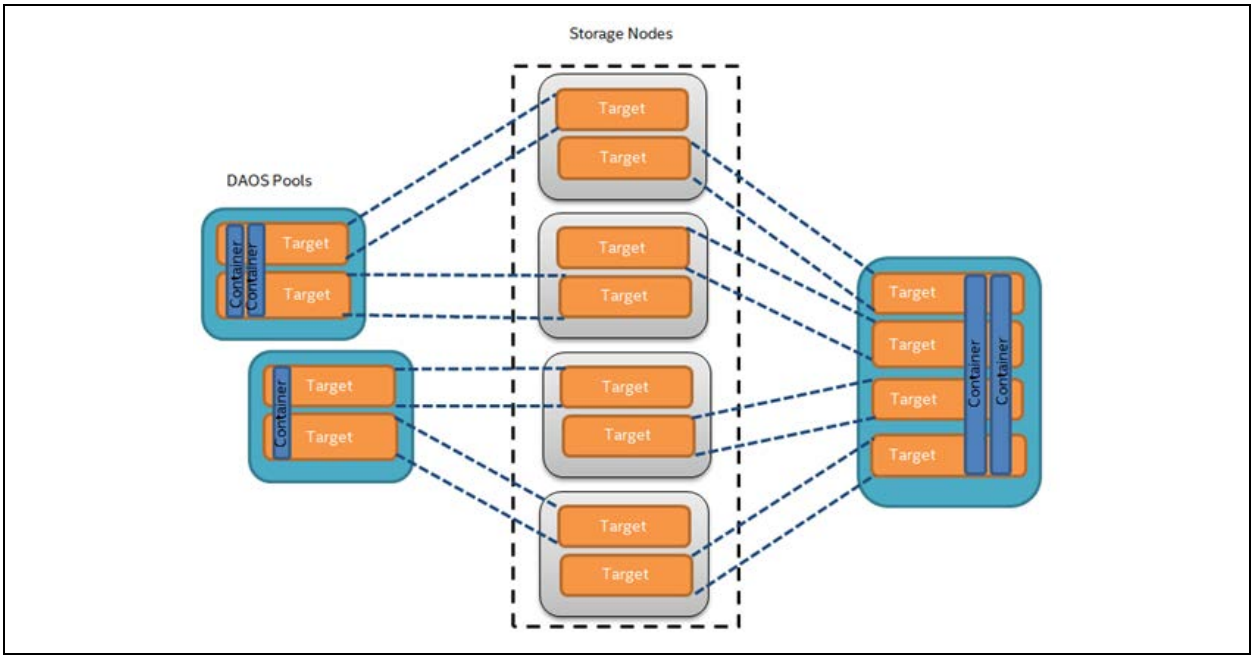


Figure 2-2. Example of four Storage Nodes, eight DAOS Targets and three DAOS Pools

Table 2-1 shows the targeted level of scalability for each DAOS abstraction.

Table 2-1, DAOS Scalability

DAOS Concept	Component Order of Magnitude Limit
System	10^2 Pools (hundreds)
Pool	10^2 Containers (hundreds)
Container	10^9 Objects (billions)

2.3.1 DAOS Pool

A Pool is identified by a unique UUID and maintains target memberships in the pool map stored in persistent memory. The pool map not only records the list of active targets, it also contains the storage topology under the form of a tree that is used to identify targets sharing common hardware components. For instance, the first level of the tree can represent targets sharing the same motherboard, then the second level can represent all motherboards sharing the same rack and finally the third level can represent all racks in the same cage. This framework effectively represents hierarchical fault domains, which are then used to avoid placing redundant data on targets subject to correlated failures. At any point in time, new targets can be added to the pool map and failed ones can be excluded. Moreover, the pool map is fully versioned, which effectively assigns a unique sequence to each modification of the map, more particularly for failed node removal.

A pool shard is a reservation of NVM storage (i.e. SCM optionally combined with a pre-allocated space on NVMe storage) on a specific target. It has a fixed capacity and fails operations when full. Current space usage can be queried at any time and reports the total amount of bytes used by any data type stored in the pool shard. Space consumed on the different type of storage is reported separately.

Upon target failure and exclusion from the pool map, data redundancy inside the pool is automatically restored while the pool remains online. Rebuild progress is recorded regularly in special logs in the pool stored in persistent memory to address cascading failures. When new targets are added, data is automatically migrated to the newly added targets to redistribute space usage equally among all the members. This process is known as space rebalancing and uses dedicated persistent logs as well to support interruption and restart. A pool is a set of targets spread across different storage nodes over which data and metadata are distributed to achieve horizontal scalability, and replicated or erasure-coded to ensure durability and availability.

When creating a pool, a set of system properties must be defined to configure the different features supported by the pool. In addition, user can define their own attributes that will be stored persistently.

A pool is only accessible to authenticated and authorized applications. Multiple security frameworks could be supported, from NFSv4 access control lists to third party-based authentication (such as Kerberos). Security is enforced when connecting to the pool. Upon successful connection to the pool, a connection context is returned to the application process.

A pool stores many different sorts of persistent metadata, such as the pool map, authentication and authorization information, user attributes, properties and rebuild logs. Such metadata are critical and require the highest level of resiliency. Therefore, the pool metadata are replicated on a few nodes from distinct high-level fault domains. For very large configurations with hundreds of thousands of storage nodes, only a very small fraction of those nodes (in the order of tens) run the pool metadata service. With a limited number of storage nodes, DAOS can afford to rely on a consensus algorithm to reach agreement and to guarantee consistency in the presence of faults and to avoid split-brain syndrome.

2.3.2 DAOS Container

A container represents an object address space inside a pool and is identified by a UUID. Applications (i.e. directly or via I/O middleware, domain-specific data format, big data or AI

frameworks) store all related datasets into a container which is the unit of storage management for the user.

Likewise to pools, containers can store user attributes and a set of properties must be passed at container creation time to configure different features like checksums.

Objects in a container are identified by a unique 128-bit object address and may have different schemas for data distribution and redundancy over targets. Dynamic or static striping, replication or erasure code are some parameters required to define the object schema. The object class defines common schema attributes for a set of objects. Each object class is assigned a unique identifier and is associated with a given schema at the pool level. A new object class can be defined at any time with a configurable schema, which is then immutable after creation, or at least until all objects belonging to the class have been destroyed. For convenience, several object classes expected to be the most commonly used will be predefined by default when the pool is created, as shown in Table 2-2.

Table 2-2. Sample of Pre-defined Object Classes

Object Class (RW = read/write, RM = read-mostly	Redundancy	Metadata in OIT, (SC = stripe count, RC = replica count, PC = parity count, TGT = target
Small size & RW	Replication	No (static SCxRC, e.g. 1x4)
Small size & RM	Erasure code	No (static SC+PC, e.g. 4+2)
Large size & RW	Replication	No (static SCxRC over max #targets)
Large size & RM	Erasure code	No (static SCx(SC+PC) w/ max #TGT)
Unknown size & RW	Replication	SCxRC (e.g. 1x4 initially and grows)
Unknown size & RM	Erasure code	SC+PC (e.g. 4+2 initially and grows)

A container is the unit of transaction and snapshot. Container metadata (i.e. list of snapshots, container open handles, object class, user attributes, properties, etc.) are stored in persistent memory and maintained by a dedicated container metadata service that either uses the same replicated engine as the parent metadata pool service, or has its own engine.

2.3.3 DAOS Object

To avoid scaling problems and overhead common to traditional storage system, DAOS objects are intentionally simple. No default object metadata beyond the type and schema are provided. This means that the system does not maintain time, size, owner, permissions or even track openers. To achieve high availability and horizontal scalability, many object schemas (replication/erasure code, static/dynamic striping, etc.) are provided. The schema framework is flexible and easily expandable to allow for new custom schema types in the future. The layout is generated algorithmically on object open from the object identifier and the pool map. End-to-end integrity is assured by protecting object data with checksums during network transfer and storage.

A DAOS object can be accessed through different native interfaces exported by libdaos: multi-level key-array, key-value or array APIs that allows to represent efficiently structured, semi-structured or unstructured data.

3 Hardware Requirements

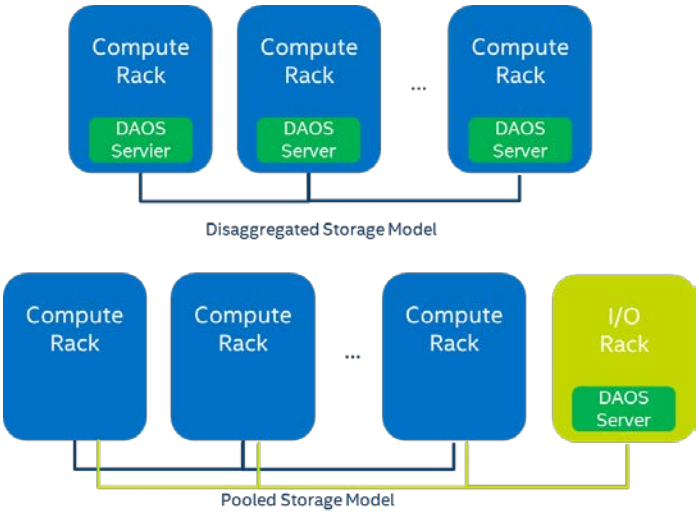
The purpose of this section is to describe processor, storage and network requirements to deploy a DAOS system.

3.1 Deployment Options

As illustrated in the figure below, a DAOS system can be deployed in two different ways:

- **Pooled Storage Model** : The DAOS servers can run on dedicated storage nodes in separate racks. This is a traditional pool model where storage is uniformly accessed by all compute nodes. In order to minimize the number of I/O racks and to optimize floor space, this approach usually requires high density storage servers.
- **Disaggregated Storage Model** : In the disaggregated model, the storage nodes are integrated into compute racks and can be either dedicated or shared (e.g. in a hyper-converged infrastructure) nodes. The DAOS servers are thus massively distributed and storage access is non-uniform and must take locality into account.

While DAOS is mostly deployed following the pooled model, active research is conducted to efficiently support the disaggregated model as well.



3.2 Processor Requirements

DAOS requires a 64-bit processor architecture and is primarily developed on Intel 64 architecture. The DAOS software and the libraries it depends on (e.g. ISA-L, SPDK, PMDK and DPDK) can take advantage of Intel® SSE and AVX extensions.

DAOS is also regularly tested on 64-bit ARM processors configured in Little Endian mode. The same build instructions that are used for x86-64 are applicable for ARM builds as well. DAOS and its dependencies will make the necessary adjustments automatically in their respective build systems for ARM platforms.

3.3 Network Requirements

The DAOS network layer relies on libfabrics and supports OFI providers for Ethernet/sockets, InfiniBand/verbs, RoCE, Cray's GNI, and the Intel Omni-Path Architecture.

An RDMA-capable fabric is preferred for better performance. DAOS can support multiple rails by binding different instances of the DAOS server to individual network card.

An additional out-of-band network connecting the nodes in the DAOS service cluster is required for DAOS administration. Management traffic uses IP over Fabric.

3.4 Storage Requirements

DAOS requires each storage node to have direct access to storage-class memory (SCM). While DAOS is primarily tested and tuned for Optane DC Persistent Memory, the DAOS software stack is built over the Persistent Memory Development Kit (PMDK) and the DAX feature of the Linux and Windows operating systems as described in the SNIA NVM Programming Model[4]. As a result, the open-source DAOS software stack should be able to run transparently over any type of storage-class memory supported by PMDK.

The storage node can be optionally equipped with NVMe (non-volatile memory express) SSDs to provide capacity. HDDs as well as SATA and SAS SSDs are not supported by DAOS. Both NVMe 3D-NAND and Optane SSDs are supported. Optane SSDs are preferred for DAOS installation that targets a very high IOPS rate. NVMe-oF devices are also supported by the userspace storage stack, but have never been tested.

The minimal recommended ratio between SCM and SSDs capacity is 6% to guarantee that DAOS has enough space in SCM to store internal metadata (e.g. pool metadata, SSD block allocation tracking).

For testing purposes, SCM can be emulated with DRAM by mounting a tmpfs filesystem and NVMe SSDs can be also emulated with DRAM or a loopback file.

3.5 CPU Affinity

On recent Xeon platforms, PCIe slots have a natural affinity to one CPU. Although globally accessible from any of the system cores, NVMe SSDs and network interface cards connected through the PCIe bus may provide different performance characteristics (e.g., higher latency, lower bandwidth) to each CPU. Accessing “remote” PCIe devices may involve traffic over the UPI (Ultra Path Interconnect) link that might become a point of congestion. Similarly, persistent memory is non-uniformly accessible (NUMA), and CPU affinity must be respected for maximal performance.

Therefore, when running in a multi-socket and multi-rail environment, the DAOS service must be able to detect the CPU to PCIe device and persistent memory affinity and minimize as much as possible non-local access. This can be achieved by spawning one instance of the I/O server per CPU, then accessing only local persistent memory and PCI devices from that server instance. The control plane is responsible for detecting the storage and network affinity and starting the I/O servers accordingly.

3.6 Fault Domains

DAOS relies on single-ported storage massively distributed across different storage nodes. Each storage node is thus a single point of failure. DAOS achieves fault tolerance by providing data redundancy across storage nodes in different fault domains.

[4] https://www.snia.org/sites/default/files/technical_work/final/NVMProgrammingModel_v1.2.pdf

DAOS assumes that fault domains are hierarchical and do no overlap. For instance, the first level of fault domain could be the racks and the second one the storage nodes.

For efficient placement and optimal data resilience, the more fault domains, the better. As a result, it is preferable to distribute storage nodes across as many racks as possible.

4 DAOS Software Installation

DAOS runs on both Intel 64 and ARM64 platforms and has been successfully tested on CentOS7, OpenSUSE 42.2 and Ubuntu 18.04 distributions.

4.1 Software Dependencies

DAOS requires a C99-capable compiler, a golang compiler, and the scons build tool. Moreover, the DAOS stack leverages the following open source projects:

- CaRT for rank-based transport services that rely on both Mercury and Libfabric for lightweight network transport and PMIx for process set management. See the CaRT repository for more information on how to build the CaRT library.
- PMDK for persistent memory programming.
- SPDK for userspace NVMe device access and management.
- FIO for flexible testing of Linux I/O subsystems, specifically enabling validation of userspace NVMe device performance through fio-spdk plugin.
- ISA-L for checksum and erasure code computation.
- Argobots for thread management.

The DAOS build system can be configured to download and build any missing dependencies automatically.

4.2 Distribution Packages

DAOS RPM packaging is under development and will be available for DAOS v1.0. Integration with the Spack package manager is also under consideration.

4.3 DAOS Source Code

To check out the DAOS source code, run the following command:

```
git clone https://github.com/daos-stack/daos.git
```

This command clones the DAOS git repository (path referred as ${daospath} below). Then initialize the submodules with:

```
cd ${daospath}

git submodule init

git submodule update
```

4.4 Building DAOS from Scratch

The below instructions have been verified with CentOS. Installations on other Linux distributions might be similar with some variations. Developers of DAOS may want to check additional sections below before beginning for suggestions related specifically to development. Please contact us in our forum if running into issues.

4.4.1 Build Prerequisites

Please install the following software packages (or equivalent for other distros):

On CentOS and OpenSUSE:

```
yum install -y epel-release
yum install -y git gcc gcc-c++ make cmake golang libtool scons boost-devel
yum install -y libuuid-devel openssl-devel libevent-devel libtool-ltdl-devel
yum install -y librdmacm-devel libcmocka libcmocka-devel readline-devel
yum install -y doxygen pandoc flex patch nasm yasm
yum install -y ninja-build meson libyaml-devel
# Required SPDK packages for managing NVMe SSDs
yum install -y CUnit-devel libaio-devel astyle-devel python-pep8 lcov
yum install -y python clang-analyzer sg3_utils libiscsi-devel
yum install -y libibverbs-devel numactl-devel doxygen mscgen graphviz
# Required IpmCtl packages for managing SCM Modules
yum install -y yum-plugin-copr epel-release
yum copr -y enable jhli/ipmctl
yum copr -y enable jhli/safeclib
yum install -y libipmctl-devel
```

On Ubuntu and Debian:

```
apt-get install -y git gcc golang make cmake libtool-bin scons autoconf
apt-get install -y libboost-dev uuid-dev libssl-dev libevent-dev libltdl-dev
apt-get install -y librdmacm-dev libcmocka0 libcmocka-dev libreadline6-dev
apt-get install -y curl doxygen pandoc flex patch nasm yasm
apt-get install -y ninja-build meson libyaml-dev python2.7-dev
# Required SPDK packages for managing NVMe SSDs
apt-get install -y libibverbs-dev librdmacm-dev libcunit1-dev graphviz
apt-get install -y libaio-dev sg3-utils libiscsi-dev doxygen mscgen libnuma-dev
# Required IpmCtl packages for managing SCM Modules
apt-get install -y software-properties-common
add-apt-repository ppa:jhli/libsafec
add-apt-repository ppa:jhli/ipmctl
apt-get update
apt-get install -y libipmctl-dev
```

Verify that all the auto tools listed below are at the appropriate versions:

- m4 (GNU M4) 1.4.16
- flex 2.5.37
- autoconf (GNU Autoconf) 2.69
- automake (GNU automake) 1.13.4
- libtool (GNU libtool) 2.4.2

4.4.2 Protobuf Compiler

The DAOS control plane infrastructure will be using protobuf as the data serialization format for its RPC requests. The DAOS proto files use protobuf 3 syntax which is not supported by the platform protobuf compiler in all cases. Not all developers will need to build the proto files into the various source files. However, if changes are made to the proto files, they will need to be regenerated with a protobuf 3.* or higher compiler. To set up support for compiling protobuf files, download the following precompiled package for Linux and install it somewhere accessible by your PATH variable.

`https://github.com/google/protobuf/releases/download/v3.5.1/protoc-3.5.1-linux-x86_64.zip`

4.4.3 Building DAOS & Dependencies

If all the software dependencies listed previously are already satisfied, then type the following command in the top source directory to build the DAOS stack:

```
scons --config=force install
```

If you are a developer of DAOS, we recommend following the instructions in Section 4.4.4 below.

Otherwise, the missing dependencies can be built automatically by invoking scons with the following parameters:

```
scons --config=force --build-deps=yes USE_INSTALLED=all install
```

By default, DAOS and its dependencies are installed under ${daospath}/install. The installation path can be modified by adding the PREFIX= option to the above command line (e.g., PREFIX=/usr/local).

4.4.4 Environment setup

Once built, the environment must be modified to search for binaries and header files in the installation path. This step is not required if standard locations (e.g. /bin, /sbin, /usr/lib, ...) are used.

```
CPATH=${daospath}/install/include/:$CPATH

PATH=${daospath}/install/bin/:${daospath}/install/sbin:$PATH

export CPATH PATH
```

If using bash, PATH can be set up for you after a build by sourcing the script scons_local/utils/setup_local.sh from the daos root. This script utilizes a file generated by the build to determine the location of daos and its dependencies.

If required, ${daospath}/install must be replaced with the alternative path specified through PREFIX. The network type to use as well the debug log location can be selected as follows:

```
export CRT_PHY_ADDR_STR="ofi+sockets",

OFI_INTERFACE=eth0, where eth0 is the network device you want to use.
```

For infiniband you could use ib0 or whichever label points to IB device.

5 DAOS System Deployment

5.1 Preflight Checklist

This section covers the preliminary setup required on the compute and storage nodes before deploying DAOS.

5.1.1 Time Synchronization

The DAOS transaction model relies on timestamps and requires time to be synchronized across all the storage and client nodes. This can be done via NTP or any other equivalent protocol.

5.1.2 User & Group

DAOS requires users and groups to be synchronized on both storage and client nodes.

5.1.3 Runtime Directory Setup

DAOS uses a series of Unix Domain Sockets to communicate between its various components. On modern Linux systems Unix Domain Sockets are typically stored under /run or /var/run (usually a symlink to /run) and are a mounted tmpfs file system. There are several methods for ensuring the necessary directories are setup.

A sign that this step may have been missed is when starting daos_server or daos_agent you may see the message:

```
mkdir /var/run/daos_server: permission denied

Unable to create socket directory: /var/run/daos_server
```

5.1.3.1 Non-default Directory

By default daos_server and daos_agent will use the directories /var/run/daos_server and /var/run/daos_agent respectively. To change the default location that daos_server uses for its runtime directory either uncomment and set the socket_dir configuration value in install/etc/daos_server.yaml or pass the location to daos_server on the command line using the `-d` flag. For the daos_agent an alternate location can be passed on the command line using the `-runtime_dir` flag.

5.1.3.2 Default Directory (non-persistent)

Files and directories created in /run and /var/run only survive until the next reboot. However if reboots are infrequent an easy solution while still utilizing the default locations is to manually create the required directories. To do this execute the following commands.

daos_server:

- `mkdir /var/run/daos_server`
- `chmod 0755 /var/run/daos_server`
- `chown user:user /var/run/daos_server` (where user is the user you will run daos_server as)

daos_agent:

- `mkdir /var/run/daos_agent`

- `chmod 0755 /var/run/daos_agent`
- `chown user:user /var/run/daos_agent` (where user is the user you will run daos_agent as)

5.1.3.3 Default Directory (persistent)

If the server hosting daos_server or daos_agent will be rebooted often systemd provides a persistent mechanism for creating the required directories called tmpfiles.d. This mechanism will be required every time the system is provisioned and requires a reboot to take effect.

To tell systemd to create the necessary directories for DAOS:

- Copy the file utils/systemd/daosfiles.conf to /etc/tmpfiles.d
  `cp utils/systemd/daosfiles.conf /etc/tmpfiles.d`
- Modify the copied file to change the user and group fields (currently daos) to the user daos will be run as
- Reboot the system and the directories will be created automatically on all subsequent reboots.

5.1.4 Root Privilege Access

Several tasks (e.g. storage access, hugepages configuration) performed by the DAOS server requires elevated permissions on the storage nodes.

NVMe access through SPDK as an unprivileged user can be enabled by first running sudo daos_server prep-nvme -p 4096 -u bob. This will perform the required setup in order for daos_server to be run by user "bob" who will own the hugepage mountpoint directory and vfio groups as needed in SPDK operations. If the target-user is unspecified (-u short option), the target user will be the issuer of the sudo command (or root if not using sudo). The specification of hugepages (-p short option) defines the number of huge pages to allocate for use by SPDK.

The configuration commands that require elevated permissions are in src/control/mgmt/init/setup_spdk.sh (script is installed as install/share/setup_spdk.sh).

The sudoers file can be accessed with command visudo and permissions can be granted to a user to execute a specific command pattern (requires prior knowledge of daos_server binary location):

```
linuxuser ALL=/home/linuxuser/projects/daos_m/install/bin/daos_server prep-nvme*
```

See daos_server prep-nvme --help for usage.

5.1.5 Storage Detection & Selection

While the DAOS server will eventually auto-detect all the usable storage, the administrator will still be provided the ability through the configuration file (see next section) to whitelist or blacklist the storage devices to be (or not) used. This section covers how to manually detect the storage devices potentially usable by DAOS in order to populate the configuration file when the administrator wants to have finer control over the storage selection.

5.1.5.1 SCM

This section addresses how to verify that Optane DC Persistent memory (DCPM) is correctly installed on the storage nodes and how to configure it in interleaved mode to be used by DAOS in AppDirect mode. Instructions for other type of SCM may be covered in the future.

DCPM can be configured and managed through the IpmCtl[5] library and associated tool. The ipmctl command just be run as root and has pretty detailed man pages and help output (use “ipmctl help” to display it).

The list of NVDIMMs can be displayed as follows:

```
$ ipmctl show -dimm
 DimmID | Capacity  | HealthState | ActionRequired | LockState | FWVersion
==============================================================================
 0x0001 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x0101 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x1001 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x1101 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
```

As for affinity to CPU, use the following command:

```
 # ipmctl show -dimm
 DimmID | Capacity  | HealthState | ActionRequired | LockState | FWVersion
==============================================================================
 0x0001 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x0101 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x1001 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
 0x1101 | 502.5 GiB | Healthy     | 0              | Disabled  | 01.00.00.5127
```

Moreover, DAOS requires DCPM to be configured in interleaved mode. A command mode option (--set-interleaved) can be used as a "one-shot" invocation of *daos_server* and must be run as root. SCM modules will be configured into interleaved regions with memory mode set to "app-direct" with one set per socket (each module is assigned to socket and reports this via its NUMA rating). This configuration may require a reboot and *daos_server* will exit on completion of the task.

This can be done manually via the following commands:

```
# to verify there is non-volatile memory type
$ ipmctl show -a -topology | egrep 'Capac|MemoryType'
  MemoryType=DCPM
  Capacity=502.5 GiB
  MemoryType=DCPM
  Capacity=502.5 GiB
  MemoryType=DCPM
  Capacity=502.5 GiB
  […]
$ ipmctl create -goal PersistentMemoryType=AppDirect
```

A reboot is required after those changes.

[5] https://github.com/intel/ipmctl

5.1.5.2 SSDs

DAOS supports only NVMe-capable SSDs that are accessed directly from userspace through the SPDK library.

In order to make the SSDs available to SPDK, the setup script is run to unbind the devices from original kernel drivers and then bind the devices to a generic driver through which SPDK can communicate. The devices are then bound back to the original drivers when management activities have been completed in the control plane.

A command mode option (--show-storage) can be used as a "one-shot" invocation of *daos_server* and must be run as root to display all the locally attached SSDs (and also SCM modules) usable by DAOS.

```
$ daos_server show-storage
[…]
Listing attached storage...
NVMe:
- id: 0
  model: 'INTEL SSDPED1K375GA '
  serial: 'PHKS7335006W375AGN  '
  pciaddr: 0000:81:00.0
  fwrev: E2010420
  namespace:
  - id: 1
    capacity: 375
```

The pciaddr field above is what should be used in the server configuration file to identified NVMe SSDs.

5.1.6 Network Interface Detection & Selection

To display the supported OFI provider, use the following command:

```
# /scratch/standan/daos_m/opt/ofi/bin/fi_info -l
psm2:
    version: 1.7
ofi_rxm:
    version: 1.0
ofi_rxd:
    version: 1.0
verbs:
    version: 1.0
UDP:
    version: 1.1
sockets:
    version: 2.0
tcp:
    version: 0.1
```

```
ofi_perf_hook:
    version: 1.0
ofi_noop_hook:
    version: 1.0
shm:
    version: 1.0
ofi_mrail:
    version: 1.0
```

The fi_pingpong test (delivered as part of OFI/libfabric) can be used to verify that the targeted OFI provider works fine:

```
node1$ fi_pingpong -p psm2
node2$ fi_pingpong -p psm2 ${IP_ADDRESS_NODE1}
bytes   #sent   #ack     total       time     MB/sec    usec/xfer   Mxfers/sec
64      10      =10      1.2k        0.00s     21.69       2.95       0.34
256     10      =10      5k          0.00s    116.36       2.20       0.45
1k      10      =10      20k         0.00s    379.26       2.70       0.37
4k      10      =10      80k         0.00s   1077.89       3.80       0.26
64k     10      =10      1.2m        0.00s   2145.20      30.55       0.03
1m      10      =10      20m         0.00s   8867.45     118.25       0.01
```

Further details will be added to this section in a future revision.

5.2 Server Configuration

This section addresses how to configure the DAOS servers on the storage nodes before starting it.

5.2.1 Certificate Generation

The DAOS security framework relies on certificates to authenticate administrators. The security infrastructure is currently under development and will be delivered in DAOS v1.0.

5.2.2 Server Configuration File

The daos_server configuration file is parsed when starting the daos_server process. The configuration file location can be specified on the command line (daos_server -h for usage) or default location (install/etc/daos_server.yml).

Parameters will be parsed and populated with defaults (located in config_types.go), if not present in the configuration.

Command line parameters take precedence over configuration file values. If not specified on the command line, configuration file values will be applied (or parsed defaults).

For convenience, either active parsed config values are written to the directory where the configuration file was read from or /tmp/ if that fails.

An example of the configuration file with descriptions is provided in the GitHub source[6].

[6] https://github.com/daos-stack/daos/tree/master/utils/config

If the user shell executing the daos_server has environment variable CRT_PHY_ADDR_STR set, the user os environment will be used instead of the configuration file. In this situation a "Warning: using os env vars..." message will be printed to the console and no environment variables will be added as specified in the env_vars list within the per-server section of the server config file. This behavior provides backward compatibility with historic mechanism of specifying all parameters through environment variables.

The following section lists the format, options, defaults and descriptions available in the configuration file.

5.2.2.1 Configuration File Options

This section lists the default empty configuration listing all the options (living documentation of the config file). Live examples are available at https://github.com/daos-stack/daos/tree/master/utils/config

The Location of this configuration file is determined by first checking for the path specified through the -f option of the daos_server command line. Otherwise, /etc/daos_server.conf is used.

Name associated with the DAOS system

Immutable after reformat.

name: daos

Access points

To operate, DAOS will need a quorum of access point nodes to be available.

Immutable after reformat.

Hosts can be specified with or without port, default port below assumed if not specified.

default: hostname of this node at port 10000 for local testing

access_points: ['hostname1:10001','hostname2:10001','hostname3:10001']

access_points: [hostname1,hostname2,hostname3]

Force default port

Force different port number to bind daos_server to, this will also be used when connecting to access points if no port is specified.

default: 10000

port: 10001

Path to CA certificate

If not specified, DAOS will start in insecure way which means that anybody can administrate the DAOS installation and access data.

ca_cert: ./.daos/ca.crt

Path to server certificate and key file

Discarded if no CA certificate is passed.

default: ./.daos/daos_server.{crt,key}

cert: ./.daosa/daos_server.crt

key: ./.daosa/daos_server.key

Fault domain path

Immutable after reformat.

default: /hostname for a local configuration w/o fault domain

fault_path: /vcdu0/rack1/hostname

Fault domain callback

Path to executable which will return fault domain string.

Immutable after reformat.

fault_cb: ./.daos/fd_callback

Use specific OFI interfaces

Specify either a single fabric interface that will be used by all spawned servers or a comma-separated list of fabric interfaces to be assigned individually.

By default, the DAOS server will auto-detect and use all fabric interfaces if any and fall back to socket on the first eth card, otherwise.

fabric_ifaces: [qib0,qib1]

Use specific OFI provider

Force a specific provider to be used by all the servers.

The default provider depends on the interfaces that will be auto-detected:

ofi+psm2 for Omni-Path, ofi+verbs;ofi_rxm for Infiniband/RoCE and finally

ofi+socket for non-RDMA-capable Ethernet.

provider: ofi+verbs;ofi_rxm

Storage mount directory

TODO: If no pre-configured mountpoins are specified, DAOS will auto-detect NVDIMMs, configure them in interleave mode, format with ext4 and mount with the DAX extension creating a subdirectory within scm_mount_path.

This option allows to specify a preferred path where the mountpoints will be created. Either the specified directory or its parent must be a mount point.

default: /mnt/daos

scm_mount_path: /mnt/daosa

NVMe SSD whitelist

Only use NVMe controllers with specific PCI addresses.

Immutable after reformat, colons replaced by dots in PCI identifiers.

By default, DAOS will use all the NVMe-capable SSDs that don't have active mount points.

bdev_include: ["0000:81:00.1","0000:81:00.2","0000:81:00.3"]

NVMe SSD blacklist

Only use NVMe controllers with specific PCI addresses. Overrides drives listed in nvme_include and forces auto-detection to skip those drives.

Immutable after reformat, colons replaced by dots in PCI identifiers.

bdev_exclude: ["0000:81:00.1"]

Use Hyperthreads

When Hyperthreading is enabled and supported on the system, this parameter defines whether the DAOS service thread should only be bound to different physical cores (value 0) or hyperthreads (value 1).

default: false

hyperthreads: true

Use the given directory for creating unix domain sockets

DAOS Agent and DAOS Server both use unix domain sockets for communication with other system components. This setting is the base location to place the sockets in.

default: /var/run/daos_server

socket_dir: ./.daos/daos_server

Number of hugepages to allocate for use by NVMe SSDs

Specifies the number (not size) of hugepages to allocate for use by NVMe through SPDK. This indicates the total number to be used by any spawned servers. Default system hugepage size will be used and hugepages will be evenly distributed between CPU nodes.

default: 1024

nr_hugepages: 4096

Force specific debug mask for daos_server (control plane).

By default, just use the default debug mask used by daos_server.

Mask specifies minimum level of message significance to pass to logger.

Currently supported values are DEBUG and ERROR.

default: DEBUG

control_log_mask: ERROR

Force specific path for daos_server (control plane) logs.

default: print to stderr

control_log_file: /tmp/daos_control.log

When per-server definitions exist, auto-allocation of resources is not performed. Without per-server definitions, node resources will automatically be assigned to servers based on NUMA ratings, there will be a one-to-one relationship between servers and sockets.

servers:

Rank to be assigned as identifier for server.

Immutable after reformat.

Optional parameter, will be auto generated if not supplied.

rank: 0

Logical CPU assignments as identified in /proc/cpuinfo (e.g. [0-24] for CPU 0 to 24).

Immutable after reformat.

cpus: [0-20]

Use specific OFI interfaces.

Specify the fabric network interface that will be used by this server.

Optionally specify the fabric network interface port that will be used by this server but please only if you have a specific need, this will normally be chosen automatically.

fabric_iface: qib0

fabric_iface_port: 20000

Force specific debug mask (D_LOG_MASK) at start up time.

By default, just use the default debug mask used by DAOS.

Mask specifies minimum level of message significance to pass to logger.

default: ERR

log_mask: WARN

Force specific path for DAOS debug logs.

default: /tmp/daos.log

log_file: /tmp/daos_server1.log

Pass specific environment variables to the DAOS server

Empty by default.

env_vars:

ABT_MAX_NUM_XSTREAMS=100

CRT_TIMEOUT=30

Define a pre-configured mountpoint for storage class memory to be used by this server.

Path should be unique to server instance (can use different subdirs).

Either the specified directory or its parent must be a mount point.

scm_mount: /mnt/daos/1

Backend block device type. Force a SPDK driver to be used by this server instance.

Options are:

"nvme" for NVMe SSDs (preferred option)

"malloc" to emulate a NVMe SSD with memory

"file" to emulate a NVMe SSD with a regular file

"kdev" to use a kernel block device

Immutable after reformat.

default: nvme

bdev_class: nvme

Backend block device configuration to be used by this server instance.

Immutable after reformat.

When bdev_class is set to nvme, bdev_list is the list of unique NVMe IDs that should be different across different server instance.

Colons replaced by dots in PCI identifiers.

bdev_list: ["0000:81:00.0"] # generate regular nvme.conf

Rank to be assigned as identifier for server.

Immutable after reformat.

Optional parameter, will be auto generated if not supplied.

rank: 1

Logical CPU assignments as identified in /proc/cpuinfo (e.g. [0-24] for CPU 0 to 24).

Immutable after reformat.

cpus: [21-40]

Use specific OFI interfaces.

Specify the fabric network interface that will be used by this server. Optionally specify the fabric network interface port that will be used by this server but only if you have a specific need, this will normally be chosen automatically.

fabric_iface: qib0

fabric_iface_port: 20000

Force specific debug mask (D_LOG_MASK) at start up time.

By default, just use the default debug mask used by DAOS. Mask specifies minimum level of message significance to pass to logger.

default: ERR

log_mask: WARN

Force specific path for DAOS debug logs.

default: /tmp/daos.log

log_file: /tmp/daos_server2.log

Pass specific environment variables to the DAOS server

Empty by default.

env_vars:

ABT_MAX_NUM_XSTREAMS=200

CRT_TIMEOUT=100

Define a pre-configured mountpoint for storage class memory to be used by this server.

Path should be unique to server instance (can use different subdirs).

scm_mount: /mnt/daos/2

Backend block device type. Force a SPDK driver to be used by this server instance.

Options are:

"nvme" for NVMe SSDs (preferred option)

"malloc" to emulate a NVMe SSD with memory

"file" to emulate a NVMe SSD with a regular file

"kdev" to use a kernel block device

Immutable after reformat.

When bdev_class is set to malloc, bdev_number is the number of devices to allocate and bdev_size is the size in GB of each LUN/device.

bdev_class: malloc

bdev_number: 1

bdev_size: 4

When bdev_class is set to file, bdev_list is the list of file paths that will be used to emulate NVMe SSDs. The size of each file is specified by

bdev_size in GB unit.

bdev_class: file

bdev_list: [/tmp/daos-bdev1,/tmp/daos-bdev2]

bdev_size: 16

When bdev_class is set to kdev, bdev_list is the list of unique kernel block devices that should be different across different server instance.

bdev_class: kdev

bdev_list: [/dev/sdc,/dev/sdd]

5.3 Server Startup

DAOS currently relies on PMIx for server wire-up and application to server connection. As a result, the DAOS servers can only be started via orterun (part of OpenMPI). A new bootstrap procedure is under implementation and will be available for DAOS v1.0. This will remove the dependency on PMIx and will allow the DAOS servers to be started individually (e.g. independently on each storage node via systemd) or collectively (e.g. pdsh, mpirun or as a Kubernetes Pod).

5.3.1 Parallel Launcher

As stated above, only orterun(1) is currently supported.

The list of storage nodes can be specified in a host file (referred to as ${hostfile}). The DAOS server and the application can be started separately but must share a URI file (referred as ${urifile}) to connect. The ${urifile} is generated by orterun using (--report-uri filename) at the server and used at the application with (--ompi-server file:filename). Also, the DAOS server must be started with the --enable-recovery option to support server failure. See the orterun(1) man page for additional options.

To start the DAOS server, run:

```
orterun -np <num_servers> --hostfile ${hostfile} --enable-recovery --report-uri ${urifile} daos_server
```

The --enable-recovery is required for fault tolerance to guarantee that the fault of one server does not cause the others to be stopped.

Hostfile used here is the same as the ones used by Open MPI. See the mpirun documentation[7] for additional details.

The --allow-run-as-root option must be added to the command line to allow the daos_server to run with root priviledged on each storage nodes.

5.3.2 Systemd Integration

A preliminary systemd script to manage the DAOS server is available under utils/system. That being said, this startup method is not supported yet since the current DAOS version still relies on PMIx for DAOS server wireup.

5.3.3 Kubernetes Pod

DAOS service integration with Kubernetes is planned and will be supported in a future DAOS version.

5.3.4 Service Monitoring

On start-up, the daos_server will create and initialize the following components:

- gRPC server to handle requests over client API
- dRPC server to handle requests from IO servers over the UNIX domain socket
- storage subsystems for handling interactions with NVM devices
- SPDK environment using a shared memory segment identifier causing the process to act as a primary in multi-process mode. From there, the main process can respond to requests over the client API for information through the SPDK interface.

The daos_shell is a transitory tool used to exercise the management api and can be used to verify that the DAOS servers are up and running. It is to be run as a standard, unprivileged user as follows:

```
$ daos_shell -l storagenode1:10001,storagenode2:10001 storage list
```

[7] https://www.open-mpi.org/faq/?category=running#mpirun-hostfile

"storagenode" should be replaced with the actual hostname of each storage node. This command will show whether the DAOS server is properly running and initialized on each storage node. A more comprehensive and user-friendly tool built over the management API is under development. A first version will be available for DAOS v1.0.

5.4 Firmware Upgrade

Firmware on an NVMe controller can be updated from an image on local storage (initially installing from a local path on the host that is running *daos_server* but to be extended to downloading remotely from central storage location).

When the controller is selected and an update firmware task runs, controller data is accessed through an existing linked list through the binding fwupdate call and a raw command specifying firmware update with local image (specified by filepath) and slot identifier. The firmware update is followed by a hard reset on the controller.

5.5 Storage Burn-in

Burn-in testing can be performed on discovered NVMe controllers. By default this involves a 15-minute slow burn-in test with a mixed read/write workload issued by fio but test duration and load strength should be user configurable. Burn-in should run in the background to allow administrators to use the control-plane for other tasks in the meantime.

The fio repo is to be built and needs to be referenced when building the SPDK fio_plugin. The plug-in can then be run by fio to exercise the NVMe device through SPDK. Currently the output of the burn-in is displayed in the shell and control is returned to the user after completion. Future iterations may perform this as a background task.

5.6 DAOS Formatting

Distributed formatting through the DAOS servers for both SCM and SSDs is under development and will be documented there once available.

Meanwhile, SCM should be formatted manually as an ext4 filesystem and mounted with the dax option prior to start the DAOS servers on each storage node:

```
# Create a /dev/pmem* device for one NUMA node
$ ndctl create-namespace
{
  "dev":"namespace1.0",
  "mode":"fsdax",
  "map":"dev",
  "size":"2964.94 GiB (3183.58 GB)",
  "uuid":"842fc847-28e0-4bb6-8dfc-d24afdba1528",
  "raw_uuid":"dedb4b28-dc4b-4ccd-b7d1-9bd475c91264",
  "sector_size":512,
  "blockdev":"pmem1",
  "numa_node":1
}

# Show regions with 0 free space after namespace created
$ ipmctl show -a -region
SocketID | ISetID           | PersistentMemoryType | Capacity   | FreeCapacity | HealthState
```

```
=====================================================================================================
 0x0000   | 0xb1887f48651e2ccc | AppDirect           | 3012.0 GiB | 3012.0 GiB   | Healthy
 0x0001   | 0xf77a7f481e352ccc | AppDirect           | 3012.0 GiB | 0.0 GiB      | Healthy
```

```
$ mkfs.ext4 /dev/pmem1
[…]

$ mount -o dax /dev/pmem1 /mnt/daos
```

If SCM is emulated with DRAM, then a tmpfs filesystem should be mounted:

```
$ mount –t tmpfs –o size=16G tmpfs /mnt/daos
```

Replace 16G with the desired tmpfs size.

5.7 Agent Configuration

The DAOS Agent is not required in DAOS v0.4 since authentication support is not fully landed yet. Instructions on how to setup and start the agent will be provided in the next revision of this document.

5.8 System Validation

To validate that the DAOS system is properly installed, the daos_test suite can be executed:

```
orterun -np <num_clients> --hostfile ${hostfile} --ompi-server file:${urifile}
./daos_test
```

daos_test requires at least 8GB of SCM (or DRAM with tmpfs) storage on each storage node.

6 DAOS System Administration

6.1 System Monitoring

System monitoring and telemetry data will be provided as part of the control plane and will be documented in a future revision.

6.2 System Operations

6.2.1 Full Shutdown and Restart

Details on how to support proper DAOS server shutdown will be provided in future revision.

6.2.2 Fault Domain Maintenance & Reintegration

Details on how to drain an individual storage node or fault domain (e.g. rack) in preparation for maintenance activity and how to reintegrate it will be provided in future revision.

6.2.3 DAOS System Extension

Ability to add new DAOS server instances to a pre-existing DAOS system will be documented in future revision.

6.3 Fault Management

DAOS relies on massively distributed single-ported storage. Each target is thus effectively a single point of failure. DAOS achieves availability and durability of both data and metadata by providing redundancy across targets in different fault domains.

6.3.1 Fault Detection & Isolation

DAOS servers are monitored within a DAOS system through a gossip-based protocol called SWIM[8] that provides accurate, efficient and scalable server fault detection.

Storage attached to each DAOS target is monitored through periodic local health assessment. Whenever a local storage I/O error is returned to the DAOS server, an internal health check procedure will be called automatically. This procedure makes an overall health assessment by analyzing the IO error code and device SMART/Health data. If the result is negative, the target will be marked as faulty, and further I/Os to this target will be rejected and re-routed.

Once detected, the faulty target or servers (effectively a set of targets) must be excluded from each pool membership. This process is triggered either manually by the administrator or automatically (see next section for more information). Upon exclusion from the pool map, each target starts the collective rebuild process automatically to restore data redundancy. The rebuild process is designed to operate online while servers continue to process incoming I/O operations from applications.

Tools to monitor and manage rebuild are still under development.

[8] https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=1028914

6.3.2 Rebuild Throttling

The rebuild process may consume a lot of resources on each server and can be throttled to reduce impact on application performance. This current logic relies on CPU cycles on the storage nodes. By default, the rebuild process is configured to consume up to 30% of the CPU cycles, leaving the other 70% for regular I/O operations.

During rebuild process, the user can set the throttle to guarantee the rebuild will not use more resource than the user setting. The user can only set the CPU cycle for now. For example, if the user set the throttle to 50, then the rebuild will at most use 50% of CPU cycle to do rebuild job. The default rebuild throttle for CPU cycle is 30. This parameter can be changed via the daos_mgmt_set_params() API call and will be eventually available through the management tools.

6.4 Software Upgrade

Interoperability in DAOS is handled via protocol and schema versioning for persistent data structures. Further instructions on how to manage DAOS software upgrades will be provided in future revision.

6.4.1 Protocol Interoperability

Limited protocol interoperability is provided by the DAOS storage stack. Version compatibility checks will be performed to verify that:

- All targets in the same pool run the same protocol version.
- Client libraries linked with the application may be up to one protocol version older than the targets.

If a protocol version mismatch is detected among storage targets in the same pool, the entire DAOS system will fail to start up and will report failure to the control API. Similarly, the connection from clients running a protocol version incompatible with the targets will return an error.

6.4.2 Persistent Schema Compatibility and Update

The schema of persistent data structures may evolve from time to time to fix bugs, add new optimizations or support new features. To that end, the persistent data structures support schema versioning.

Upgrading the schema version will not be performed automatically and must be initiated by the administrator. A dedicated upgrade tool will be provided to upgrade the schema version to the latest one. All targets in the same pool must have the same schema version. Version checks are performed at system initialization time to enforce this constraint.

To limit the validation matrix, each new DAOS release will be published with a list of supported schema versions. To run with the new DAOS release, administrators will then need to upgrade the DAOS system to one of the supported schema versions. New pool shards will always be formatted with the latest version. This versioning schema only applies to a data structure stored in persistent memory and not to block storage that only stores user data with no metadata.

6.5 Storage Scrubbing

Support for end-to-end data integrity is planned for DAOS v1.2 and background checksum scrubbing for v2.2. Once available, those functionality will be documented here.

7 DAOS Pool Operations

A DAOS pool is a storage reservation that can span any storage nodes and is managed by the administrator. The amount of space allocated to a pool is decided at creation time and can be eventually expanded through the management interface.

7.1 Pool Creation/Destroy

A DAOS pool can be created and destroyed through the DAOS management API (see daos_mgmt.h). DAOS also provides a utility called dmg to manage storage pools from the command line.

To create a pool:

```
orterun --ompi-server file:${urifile} dmg create --size=xxG -nvme=yyT
```

This command creates a pool distributed across the DAOS servers with a target size on each server with xxGB of SCM and yyTB of NVMe storage. The UUID allocated to the newly created pool is printed to stdout (referred as ${puuid}) as well as the rank where the pool service is located (referred as ${svcl}).

The typical output of this command is as follows:

```
orterun --ompi-server file:${urifile} dmg create --size=xxG -nvme=yyT
4056fb6d-9fca-4f2d-af8a-cfd57d92a92d 1:2
```

This created a pool with UUID 4056fb6d-9fca-4f2d-af8a-cfd57d92a92d with two pool service replica on rank 1 and 2.

To destroy a pool:

```
orterun --ompi-server file:${urifile} dmg destroy --pool=${puuid}
```

7.2 Pool Properties

At creation time, a list of pool properties can be specified through the API (not supported by the tool yet):

- DAOS_PROP_CO_LABEL is a string that the administrator can associated with a pool. e.g. “project A”, “project B”, “IO500 test pool”
- DAOS_PROP_PO_ACL are access control list (ACL) associated with the pool
- DAOS_PROP_PO_SPACE_RB is the space to be reserved on each target for rebuild purpose.
- DAOS_PROP_PO_SELF_HEAL defines whether the pool wants automatically-trigger or manually-triggered self-healing.
- DAOS_PROP_PO_RECLAIM is used to tune the space reclaim strategy based on time interval, batched commits or snapshot creation.

While those pool properties are currently stored persistently with pool metadata, many of them are still under development. Moreover, the ability to modify some of those properties on an existing pool will also be eventually provided.

7.3 Pool ACLs

Support for per-pool Access Control Lists (ACLs) is under development and is scheduled for DAOS v1.0. DAOS ACLs will implement a subset of the NFSv4 ACL standard. This feature will be documented here once available.

The pool query operation retrieves information (i.e. number of targets, space usage, rebuild status, property list, …) about a created pool. It is integrated into the dmg utility.

To query a pool:

```
orterun --ompi-server file:${urifile} dmg query --svc=${svcl} --pool=${puuid}
```

Below is the output for a pool created with SCM space only.

```
pool=47293abe-aa6f-4147-97f6-42a9f796d64a

Pool 47293abe-aa6f-4147-97f6-42a9f796d64a, ntarget=64, disabled=8

Pool space info:

- Target(VOS) count:56

- SCM:

  Total size: 30064771072

  Free: 30044570496, min:530139584, max:536869696, mean:536510187

- NVMe:

  Total size: 0

  Free: 0, min:0, max:0, mean:0

Rebuild done, 10 objs, 1026 recs
```

The total and free sizes are the sum across all the targets whereas min/max/mean gives information about individual targets. A min value close to 0 means that one target is running out of space.

The example below shows a rebuild in progress and NVMe space allocated.

```
pool=95886b8b-7eb8-454d-845c-fc0ae0ba5671

Pool 95886b8b-7eb8-454d-845c-fc0ae0ba5671, ntarget=64, disabled=8

Pool space info:

- Target(VOS) count:56

- SCM:

  Total size: 30064771072

  Free: 29885237632, min:493096384, max:536869696, mean:533664957

- NVMe:

  Total size: 60129542144

  Free: 29885237632, min:493096384, max:536869696, mean:533664957
```

```
Rebuild busy, 75 objs, 9722 recs
```

Additional status and telemetry data are planned to be exported through the management API and tool and will be documented here once available.

7.4 Pool Modifications

7.4.1 Target Exclusion & Self-Healing

To exclude a target from a pool:

```
orterun --ompi-server file:${urifile} dmg exclude --svc=${svcl} --
pool=${puuid} -target=${rank}
```

7.4.2 Pool Extension

7.4.2.1 Target Addition & Space Rebalancing

Support for online target addition and automatic space rebalancing is planned for DAOS v1.4 and will be documented here once available.

7.4.2.2 Pool Shard Resize

Support for quiescent pool shard resize is currently not supported and is under consideration.

7.5 Pool Catastrophic Recovery

A DAOS pool is instantiated on each target by a set of pmemobj files managed by PMDK and SPDK blobs on SSDs. Tools to verify and repair this persistent data is scheduled for DAOS v2.4 and will be documented here once available.

Meanwhile, PMDK provides a recovery tool (i.e. pmempool check) to verify and possibly repair a pmemobj file. As show in the previous section, the rebuild status can be consulted via the pool query and will be expanded with more information.

8 Application Interface and Tiering

8.1 DAOS Container Management

DAOS containers are the unit of data management for users.

8.1.1 Container Creation/Destroy

Containers can be created and destroyed through the daos_cont_create/destroy() functions exported by the DAOS API. A user tool called “daos” to manage containers is under development and will be available and documented here for DAOS v1.0.

8.1.2 Container Properties

At creation time, a list of container properties can be specified:

- DAOS_PROP_CO_LABEL is a string that a user can associate with a container. e.g. “Cat Pics” or “ResNet-50 training data”
- DAOS_PROP_CO_LAYOUT_TYPE is the container type (POSIX, MPI-IO, HDF5, …)
- DAOS_PROP_CO_LAYOUT_VER is a version of the layout that can be used by I/O middleware and application to handle interoperability.
- DAOS_PROP_CO_CSUM defines whether checksums are enabled or disabled and the checksum type used.
- DAOS_PROP_CO_REDUN_FAC is the redundancy factor that drives the minimal data protection required for objects stored in the container. e.g. RF1 means no data protection, RF3 only allows 3-way replication or erasure code N+2.
- DAOS_PROP_CO_REDUN_LVL is the fault domain level that should be used to place data redundancy information (e.g. storage nodes, racks …). This information will be eventually consumed to determine object placement.
- DAOS_PROP_CO_SNAPSHOT_MAX is the maximum number of snapshot to retain. When a new snapshot is taken and the threshold is reached, the oldest snapshot will be automatically deleted.
- DAOS_PROP_CO_ACL is the list of ACL for the container.
- DAOS_PROP_CO_COMPRESS and DAOS_PROP_CO_ENCRYPT are reserved to configure respectively compression and encryption. Those features are currently not on the roadmap.

While those properties are currently stored persistently with container metadata, many of them are still under development. The ability to modify some of these properties on an existing container will also be eventually provided.

8.1.3 Container Snapshot

Similar to container create/destroy, a container can be snapshotted through the DAOS API by calling daos_cont_create_snap(). Additional functions are provided to destroy and list container snapshots.

The API also provides the ability to subscribe to container snapshot events and to rollback the content of a container to a previous snapshot, but those operations are not yet fully implemented.

This section will be updated once the “daos” tool is available.

8.1.4 Container User Attributes

Similar to POSIX extended attributes, users can attach some metadata to each container through the daos_cont_{list/get/set}_attr() API.

8.1.5 Container ACLs

Support for per-container ACLs is scheduled for DAOS v1.2. Similar to pool ACLs, container ACLs will implement a subset of the NFSv4 ACL standard. This feature will be documented here once available.

8.2 Native Programming Interface

8.2.1 Building against the DAOS library

To build an applications or I/O middleware against the native DAOS API, include the daos.h header file in your program and link with -Ldaos. Examples are available under src/tests.

8.2.2 DAOS API Reference

libdaos is written in C and uses Doxygen comments that are added to C header files.

[TODO] Generate Doxygen document and add a link here.

8.2.3 Bindings to Different Languages

API bindings to both Python[9] and Go[10] languages are available.

8.3 POSIX Filesystem

A regular POSIX namespace can be encapsulated into a DAOS container. This capability is provided by the libdfs library that implements the file and directory abstractions over the native libdaos library. The POSIX emulation can be exposed to applications or I/O frameworks either directly (e.g. for frameworks Spark or TensorFlow or benchmark like IOR or mdtest that support different storage backend plugin) or transparently via a FUSE daemon combined optionally with an interception library to address some of the FUSE performance bottleneck by delivering full OS bypass for POSIX read/write operations.

8.3.1 libdfs

DFS stands for DAOS File System and is a library that allows a DAOS container to be accessed as a hierarchical POSIX namespace. It supports files, directories and symbolic links, but not hard links. Access permissions are inherited from the parent pool and not implemented on a per-file or per-directory basis. setuid() and setgid() programs as well as supplementary groups are currently not supported.

While libdfs can be tested from a single instance (i.e. single process or client node if used through dfuse), special care is required when the same POSIX container is mounted concurrently by multiple processes. Concurrent DFS mounts are not recommended. Support for concurrency control is under development and will be documented here once ready.

[9] https://github.com/daos-stack/daos/blob/master/src/utils/py/README.md
[10] https://godoc.org/github.com/daos-stack/go-daos/pkg/daos

8.3.2 dfuse

A simple high level fuse plugin (dfuse) is implemented to test the DFS API and functionality with existing POSIX tests and benchmarks (IOR, mdtest, etc.). The DFS fuse exposes one mountpoint as a single DFS namespace with a single pool and container. To test dfuse, the following steps need to be done:

1. Launch DAOS server(s) and create a pool as specified in the previous section. This will return a pool uuid "puuid" and service rank list "svcl"
2. Create an empty directory for the fuse mountpoint. For example let's use /tmp/dfs_test
3. Mount dfuse with the following command:
   `orterun -np 1 --ompi-server file:~/uri.txt dfuse /tmp/dfs_test -s -f -p puuid -l svcl`
   -p specifies the pool uuid and -l specifies the service rank list (from dmg).
4. Other arguments to dfuse: -r: option to destroy the container associated with the namespace when you umount. -d: prints debug messages at the fuse mount terminal
5. Now /tmp/dfs_test can be used as a POSIX file system (i.e., can run things like IOR/mdtest on it)
6. When done, unmount the file system: `fusermount -u /tmp/dfs_test`

Work is underway to rewrite the dfuse daemon against the low-level FUSE API.

8.3.3 libioil

An interception library called libioil is under development. This library will work in conjunction with dfuse and allow to intercept POSIX read(2) and write(2) and issue the I/O operations directly from the application context through libdaos and without any application changes.

Support for libioil is currently planned for DAOS v1.2.

8.4 Unified Namespace

The DAOS tier can be tightly integrated with the Lustre parallel filesystem in which DAOS containers will be represented through the Lustre namespace. This capability is under development and is scheduled for DAOS v1.2.

8.5 HPC I/O Middleware Support

Several HPC I/O middleware libraries have been ported to the native API.

8.5.1 MPI-IO

DAOS has its own MPI-IO ROM ADIO driver located in a MPICH fork on GitHub:

https://github.com/daos-stack/mpich

This driver has been submitted upstream for integration.

To build the MPI-IO driver:

- export MPI_LIB=""
- download the mpich repo from above and switch to daos_adio branch
- `./autogen.sh`

- `mkdir build; cd build`
- `../configure --prefix=dir --enable-fortran=all --enable-romio --enable-cxx --enable-g=all --enable-debuginfo --with-file-system=ufs+daos --with-daos=dir --with-cart=dir`
- `make -j8; make install`

Switch PATH and LD_LIBRARY_PATH where you want to build your client apps or libs that use MPI to the above installed MPICH. Note that the DAOS server will still need to be launched with OMPI's orterun. This is a unique situation where the server uses OMPI and the clients will be launched with MPICH.

Build any client (HDF5, ior, mpi test suites) normally with the mpicc and mpich library installed above (see child pages).

To run an example:

1. Launch DAOS server(s) and create a pool as specified in the previous section. This will return a pool uuid "puuid" and service rank list "svcl"
2. At the client side, the following environment variables need to be set:
   export PATH=/path/to/mpich/install/bin:$PATH
   export LD_LIBRARY_PATH=/path/to/mpich/install/lib:$LD_LIBRARY_PATH
   export MPI_LIB=""
   export CRT_ATTACH_INFO_PATH=/path/ (whatever was passed to daos_server -a)
   export DAOS_SINGLETON_CLI=1
3. `export DAOS_POOL=puuid; export DAOS_SVCL=svcl`
   This is just temporary till we have a better way of passing pool connect info to MPI-IO and other middleware over DAOS.
4. Run the client application or test.

Limitations to the current implementation include:

- Incorrect MPI_File_set_size and MPI_File_get_size - This will be fixed in the future when DAOS correctly supports records enumeration after punch or key query for max/min key and recx.
- Reading Holes does not return 0, but leaves the buffer untouched
  (Not sure how to fix this - might need to wait for DAOS implementation of iov_map_t to determine holes vs written bytes in the Array extent).
- No support for MPI file atomicity, preallocate, shared file pointers.
  (Those features were agreed upon as OK not to support.)

8.5.2 HDF5

A prototype version of a HDF5 DAOS connector is available. Please refer to the DAOS VOL connector user guide[11] for instructions on how to build and use it.

8.6 Spark Support

Spark integration with libdfs is under development and is scheduled for DAOS v1.0 or v1.2.

[11] https://bitbucket.hdfgroup.org/projects/HDF5VOL/repos/daos-vol/browse/README.md

8.7 Data Migration

8.7.1 Migration to/from a POSIX filesystem

A dataset mover tool is under consideration to move a snapshot of a POSIX, MPI-IO or HDF5 container to a POSIX filesystem and vice versa. The copy will be performed at the POSIX or HDF5 level. The resulting HDF5 file over the POSIX filesystem will be accessible through the native HDF5 connector with the POSIX VFD.

The first version of the mover tool is currently scheduled for DAOS v1.4.

8.7.2 Container Parking

The mover tool will also eventually support the ability to serialize and deserialize a DAOS container to a set of POSIX files that can be stored or “parked” in an external POSIX filesystem. This transformation is agnostic to the data model and container type and will retain all DAOS internal metadata.

9 DAOS Performance Tuning

This section will be expanded in a future revision.

9.1 Network Performance

Similar to the Lustre Network stack, the DAOS CART layer has the ability to validate and benchmark network communications in the same context as an application and using the same networks/tuning options as regular DAOS. The CART selftest can run against the DAOS servers in a production environment in a non-destructive manner. CART selftest supports different message sizes, bulk transfers, multiple targets and the following test scenarios:

- **Selftest client to servers** where selftest issues RPCs directly to a list of servers
- **Cross-servers** where selftest sends instructions to the different servers that will issue cross-server RPCs. This model supports a many to many communication model.

Instructions on how to run CART selftest will be provided in the next revision of this document.

9.2 Benchmarking DAOS

DAOS can be benchmarked with both IOR and mdtest through the following backends:

- native MPI-IO plugin combined with the ROMIO DAOS ADIO driver
- native HDF5 plugin combined with the HDF5 DAOS connector (under development)
- native POSIX plugin over dfuse and interception library (under development)
- a custom DFS plugin integrating mdtest & IOR directly with libfs without requiring FUSE or an interception library
- a custom DAOS plugin integrating IOR directly with the native DAOS array API.

10 DAOS Troubleshooting

10.1 DAOS Errors

DAOS has its own error numbering that starts at 1000. The most common errors are documented in the table below.

DAOS Error	Value	Description
DER_NO_PERM	1001	No permission
DER_NO_HDL	1002	Invalid handle
DER_INVAL	1003	Invalid parameters
DER_NOSPACE	1007	No space left on storage target
DER_NOSYS	1010	Function not implemented
DER_IO	2001	Generic I/O error
DER_ENOENT	2003	Entry not found
DER_KEY2BIG	2012	Key is too large
DER_IO_INVAL	2014	IO buffers can't match object extents

When an operation fails, DAOS returns a negative DER error. For a full list of errors, please check https://github.com/daos-stack/cart/blob/master/src/include/gurt/errno.h (DER_ERR_GURT_BASE is equal to 1000 and DER_ERR_DAOS_BASE is equal to 2000).

The function d_errstr() is provided in the API to convert an error number to an error message.

10.2 Debugging System

DAOS uses the debug system defined in CaRT but more specifically the GURT library. Log files for both client and server are written to "/tmp/daos.log" unless otherwise set by `D_LOG_FILE`.

10.2.1 Registered Subsystems/Facilities

The debug logging system includes a series of subsystems or facilities which define groups for related log messages (defined per source file). There are common facilities which are defined in GURT, as well as other facilities that can be defined on a per-project basis (such as those for CaRT and DAOS). `DD_SUBSYS` can be used to set which subsystems to enable logging. By default all subsystems are enabled (`"DD_SUBSYS=all"`).

- DAOS Facilities:
  common, tree, vos, client, server, rdb, pool, container, object, placement, rebuild, tier, mgmt, bio, tests

- Common Facilities (GURT):
  MISC, MEM
- CaRT Facilities:
  RPC, BULK, CORPC, GRP, LM, HG, PMIX, ST, IV

10.2.2 Priority Logging

All macros that output logs have a priority level, shown in descending order below.

- D_FATAL(fmt, ...) FATAL
- D_CRIT(fmt, ...) CRIT
- D_ERROR(fmt, ...) ERR
- D_WARN(fmt, ...) WARN
- D_NOTE(fmt, ...) NOTE
- D_INFO(fmt, ...) INFO
- D_DEBUG(mask, fmt, ...) DEBUG

The priority level that outputs to stderr is set with DD_STDERR. By default in DAOS (specific to the project), this is set to CRIT ("DD_STDERR=CRIT") meaning that all CRIT and more severe log messages will dump to stderr. This, however, is separate from the priority of logging to "/tmp/daos.log". The priority level of logging can be set with D_LOG_MASK, which by default is set to INFO ("D_LOG_MASK=INFO"), which will result in all messages excluding DEBUG messages being logged. D_LOG_MASK can also be used to specify the level of logging on a per-subsystem basis as well ("D_LOG_MASK=DEBUG,MEM=ERR").

10.2.3 Debug Masks/Streams:

DEBUG messages account for a majority of the log messages, and finer-granularity might be desired. Mask bits are set as the first argument passed in D_DEBUG(mask, ...). To accomplish this, DD_MASK can be set to enable different debug streams. Similar to facilities, there are common debug streams defined in GURT, as well as other streams that can be defined on a per-project basis (CaRT and DAOS). All debug streams are enabled by default ("DD_MASK=all").

- DAOS Debug Masks:
  - md = metadata operations
  - pl = placement operations
  - mgmt = pool management
  - epc = epoch system
  - df = durable format
  - rebuild = rebuild process
  - daos_default = (group mask) io, md, pl, and rebuild operations
- Common Debug Masks (GURT):
  - any = generic messages, no classification
  - trace = function trace, tree/hash/lru operations
  - mem = memory operations
  - net = network operations
  - io = object I/Otest = test programs

10.2.4 Common Use Cases

- Generic setup for all messages (default settings)

```
$ D_LOG_MASK=DEBUG
$ DD_SUBSYS=all
$ DD_MASK=all
```

- Disable all logs for performance tuning

```
$ D_LOG_MASK=ERR -> will only log error messages from all facilities
$ D_LOG_MASK=FATAL -> will only log system fatal messages
```

- Disable a noisy debug logging subsystem

```
$ D_LOG_MASK=DEBUG,MEM=ERR -> disables MEM facility by restricting all
logs from that facility to ERROR or higher priority only
```

- Enable a subset of facilities of interest

```
$ DD_SUBSYS=rpc,tests
$ D_LOG_MASK=DEBUG -> required to see logs for RPC and TESTS less severe
than INFO (the majority of log messages)
```

- Fine-tune the debug messages by setting a debug mask

```
$ D_LOG_MASK=DEBUG
$ DD_MASK=mgmt -> only logs DEBUG messages related to pool management
```

Refer to the DAOS Environment Variables documentation (Appendix B) for more information about the debug system environment.

10.3 Common DAOS Problems

This section to be updated in a future revision.

10.4 Bug Report

Bugs should be reported through our issue tracker[12] with a test case to reproduce the issue (when applicable) and debug logs.

[12] https://jira.hpdd.intel.com

Appendix A. DAOS Utilities & Usage Examples

This section to be updated in a future revision.

Appendix B. DAOS Environment Variables

This section lists the environment variables used by DAOS. Many of them are used for development purposes only and may be removed or changed in the future.

The description of each variable follows the following format:

- Short description
- Type
- The default behavior if not set.
- A longer description if necessary

This table defines a type:

Type	Values
BOOL	0 means false; any other value means true
BOOL2	no means false; any other value means true
BOOL3	set to empty or any value means true; unset means false
INTEGER	Non-negative decimal integer
STRING	String

B.1 Common environment variables

Environment variables in this section apply to both the server side and the client side

DAOS_IO_BYPASS

B.2 Server environment variables

Environment variables in this section only apply to the server side.

Variable	Description
VOS_CHECKSUM	Checksum algorithm used by VOS. STRING. Default to disabling checksums. The following checksum algorithms are supported: crc64 and crc32.
VOS_MEM_CLASS	Memory class used by VOS. STRING. Default to persistent memory. If the value is set to DRAM, all data is stored in volatile memory; otherwise, all data is stored in persistent memory.
RDB_ELECTION_TIMEOUT	Raft election timeout used by RDBs in milliseconds. INTEGER. Default to 7000 ms.
RDB_REQUEST_TIMEOUT	Raft request timeout used by RDBs in milliseconds. INTEGER. Default to 3000 ms.
DAOS_REBUILD	Determines whether to start rebuilds when excluding targets. BOOL2. Default to true.
DAOS_MD_CAP	Size of a metadata pmem pool/file in MBs. INTEGER. Default to 128 MB.
DAOS_START_POOL_SVC	Determines whether to start existing pool services when starting a daos_server. BOOL. Default to true.
DAOS_IMPLICIT_PURGE	Whether to aggregate unreferenced epochs. BOOL. Default to false.

Variable	Description
DAOS_PURGE_CREDITS	The number of credits for probing object trees when aggregating unreferenced epochs. INTEGER. Default to 1000.

B.3 Client

Environment variables in this section only apply to the client side.

Variable	Description
DAOS_SINGLETON_CLI	Determines whether to run in the singleton mode, in which the client does not need to be launched by orterun. BOOL. Default to false.

B.4 Debug System (Client & Server)

Variable	Description
D_LOG_FILE	DAOS debug logs (both server and client) are written to /tmp/daos.log by default. The debug location can be modified by setting this environment variable ("D_LOG_FILE=/tmp/daos_server").
DD_SUBSYS	Used to specify which subsystems to enable. DD_SUBSYS can be set to individual subsystems for finer-grained debugging ("DD_SUBSYS=vos"), multiple facilities ("DD_SUBSYS=eio,mgmt,misc,mem"), or all facilities ("DD_SUBSYS=all") which is also the default setting. If a facility is not enabled, then only ERR messages or more severe messages will print.
DD_STDERR	Used to specify the priority level to output to stderr. Options in decreasing priority level order: FATAL, CRIT, ERR, WARN, NOTE, INFO, DEBUG. By default, all CRIT and more severe DAOS messages will log to stderr ("DD_STDERR=CRIT"), and the default for CaRT/GURT is FATAL.
D_LOG_MASK	Used to specify what type/level of logging will be present for either all of the registered subsystems or a select few. Options in decreasing priority level order: FATAL, CRIT, ERR, WARN, NOTE, INFO, DEBUG. DEBUG option is used to enable all logging (debug messages as well as all higher priority level messages). Note that if D_LOG_MASK is not set, it will default to logging all messages excluding debug ("D_LOG_MASK=INFO"). EX: "D_LOG_MASK=DEBUG" This will set the logging level for all facilities to DEBUG, meaning that all debug messages, as well as higher priority messages will be logged (INFO, NOTE, WARN, ERR, CRIT, FATAL) EX: "D_LOG_MASK=DEBUG,MEM=ERR,RPC=ERR" This will set the logging level to DEBUG for all facilities except MEM & RPC (which will now only log ERR and higher priority level messages, skipping all DEBUG, INFO, NOTE & WARN messages)
DD_MASK	Used to enable different debug streams for finer-grained debug messages, essentially allowing the user to specify an area of interest to debug (possibly involving many different subsystems) as opposed to parsing through many lines of generic DEBUG messages. All debug streams will be enabled by default ("DD_MASK=all"). Single debug masks can be set ("DD_MASK=trace") or multiple masks ("DD_MASK=trace,test,mgmt"). Note that since these debug streams are strictly related to the debug log messages, DD_LOG_MASK must be set to DEBUG. Priority messages higher than DEBUG will still be logged for all facilities unless otherwise specified by D_LOG_MASK (not affected by enabling debug masks).