# DAOS Performance Tuning

This section will be expanded in a future revision.

## Network Performance

The DAOS [CaRT](https://github.com/daos-stack/daos/tree/master/src/cart) layer can validate and benchmark network communications in the same context as an application and using the same networks/tuning options as regular DAOS.

The CaRT self\_test can run against the DAOS servers in a production environment in a non-destructive manner. CaRT self\_test supports different message sizes, bulk transfers, multiple targets, and the following test scenarios:

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

### Getting DAOS CaRT self\_test

The CaRT self\_test and its tests are delivered as part of the daos\_client and daos\_tests [distribution packages](https://github.com/daos-stack/daos/blob/master/doc/admin/installation.md#distribution-packages). It can also be built from scratch.

$ git clone --recurse-submodules https://github.com/daos-stack/daos.git  
$ cd daos  
$ scons --build-deps=yes install  
$ cd install

For detailed information, please refer to the [DAOS build documentation](https://github.com/daos-stack/daos/blob/master/doc/admin/installation.md#building-daos--dependencies) section.

### Running CaRT self\_test

Instructions to run CaRT self\_test are as follows.

**Start DAOS server**

self\_test requires DAOS server to be running before attempt running self\_test. For detailed instruction on how to start DAOS server, please refer to the [server startup](https://github.com/daos-stack/daos/blob/master/doc/admin/deployment.md#server-startup) documentation.

**Dump system attachinfo**

self\_test will use the address information in daos\_server.attach\_info\_tmp file. To create such file, run the following command:

./bin/daos\_agent dump-attachinfo -o ./daos\_server.attach\_info\_tmp

**Prepare hostfile**

The list of nodes from which self\_test will run can be specified in a hostfile (referred to as ${hostfile}). Hostfile used here is the same as the ones used by OpenMPI. For additional details, please refer to the [mpirun documentation](https://www.open-mpi.org/faq/?category=running#mpirun-hostfile).

**Run CaRT self\_test**

The example below uses an Ethernet interface and Sockets provider. In the self\_test commands:

* **Selftest client to servers** - Replace the argument for --endpoint accordingly.
* **Cross-servers** - Replace the argument for --endpoint and --master-endpoint accordingly.

For example, if you have 8 servers, you would specify --endpoint 0-7:0 and --master-endpoint 0-7:0

The commands below will run self\_test benchmark using the following message sizes:

b1048576 1Mb bulk transfer Get and Put  
b1048576 0 1Mb bulk transfer Get only  
0 b1048576 1Mb bulk transfer Put only  
I2048 2Kb iovec Input and Output  
i2048 0 2Kb iovec Input only  
0 i2048 2Kb iovec Output only

For a full description of self\_test usage, run:

$ ./bin/self\_test --help

**To run self\_test in client-to-servers mode:**

$ /usr/lib64/openmpi3/bin/orterun --mca btl self,tcp -N 1 \  
 --hostfile ${hostfile} --output-filename testLogs/ \  
 -x D\_LOG\_FILE=testLogs/self\_test.log -x D\_LOG\_FILE\_APPEND\_PID=1 \  
 -x D\_LOG\_MASK=WARN -x CRT\_PHY\_ADDR\_STR=ofi+sockets -x OFI\_INTERFACE=eth0 \  
 -x CRT\_CTX\_SHARE\_ADDR=0 -x CRT\_CTX\_NUM=16 \  
 ./bin/self\_test --group-name daos\_server --endpoint 0-<MAX\_SERVER-1>:0 \  
 --message-sizes "b1048576,b1048576 0,0 b1048576,i2048,i2048 0,0 i2048" \  
 --max-inflight-rpcs 16 --repetitions 100 -t -n -p .

**To run self\_test in cross-servers mode:**

$ /usr/lib64/openmpi3/bin/orterun --mca btl self,tcp -N 1 \  
 --hostfile ${hostfile} --output-filename testLogs/ \  
 -x D\_LOG\_FILE=testLogs/self\_test.log -x D\_LOG\_FILE\_APPEND\_PID=1 \  
 -x D\_LOG\_MASK=WARN -x CRT\_PHY\_ADDR\_STR=ofi+sockets -x OFI\_INTERFACE=eth0 \  
 -x CRT\_CTX\_SHARE\_ADDR=0 -x CRT\_CTX\_NUM=16 \  
 ./bin/self\_test --group-name daos\_server --endpoint 0-<MAX\_SERVER-1>:0 \  
 --master-endpoint 0-<MAX\_SERVER-1>:0 \  
 --message-sizes "b1048576,b1048576 0,0 b1048576,i2048,i2048 0,0 i2048" \  
 --max-inflight-rpcs 16 --repetitions 100 -t -n -p .

## Benchmarking DAOS

DAOS can be benchmarked using several widely used IO benchmarks like IOR, mdtest, and FIO. There are several backends that can be used with those benchmarks.

### ior

IOR (<https://github.com/hpc/ior>) with the following backends:

* The IOR APIs POSIX, MPIIO and HDF5 can be used with DAOS POSIX containers that are accessed over dfuse. This works without or with the I/O interception library (libioil). Performance is significantly better when using libioil.
* A custom DFS (DAOS File System) plugin for DAOS can be used by building IOR with DAOS support, and selecting API=DFS. This integrates IOR directly with the DAOS File System (libdfs), without requiring FUSE or an interception library.
* When using the IOR API=MPIIO, the ROMIO ADIO driver for DAOS can be used by providing the daos:// prefix to the filename. This ADIO driver bypasses dfuse and directly invkes the libdfs calls to perform I/O to a DAOS POSIX container. The DAOS-enabled MPIIO driver is available in the upstream MPICH repository (MPICH 3.4.1 or higher).
* An HDF5 VOL connector for DAOS is under development. This maps the HDF5 data model directly to the DAOS data model, and works in conjunction with DAOS containers of --type=HDF5 (in contrast to DAOS container of --type=POSIX that are used for the other IOR APIs).

### mdtest

mdtest is released in the same repository as IOR. The corresponding backends that are listed above support mdtest, except for the MPI-IO and HDF5 backends that were only designed to support IOR.

### FIO

FIO can also be used to benchmark DAOS performance using dfuse and the interception library with all the POSIX based engines like sync and libaio. We do, however, provide a native DFS engine for FIO similar to what we do for IOR. That engine is available on GitHub: <https://github.com/daos-stack/dfio>

### daos\_perf

Finally, DAOS provides a tool called daos\_perf which allows benchmarking to the DAOS object API directly or to the internal VOS API, which bypasses the client and network stack and reports performance accessing the storage directly using VOS. For a full description of daos\_perf usage, run:

$ daos\_perf --help

## Client Performance Tuning

For best performance, a DAOS client should specifically bind itself to a NUMA node instead of leaving core allocation and memory binding to chance. This allows the DAOS Agent to detect the client’s NUMA affinity from its PID and automatically assign a network interface with a matching NUMA node. The network interface provided in the GetAttachInfo response is used to initialize CaRT.

To override the automatically assigned interface, the client should set the environment variable OFI\_INTERFACE to match the desired network interface.

The DAOS Agent scans the client machine on the first GetAttachInfo request to determine the set of network interfaces available that support the DAOS Server’s OFI provider. This request occurs as part of the initialization sequence in the libdaos daos\_init() performed by each client.

Upon receipt, the Agent populates a cache of responses indexed by NUMA affinity. Provided a client application has bound itself to a specific NUMA node and that NUMA node has a network device associated with it, the DAOS Agent will provide a GetAttachInfo response with a network interface corresponding to the client’s NUMA node.

When more than one appropriate network interface exists per NUMA node, the agent uses a round-robin resource allocation scheme to load balance the responses for that NUMA node.

If a client is bound to a NUMA node that has no matching network interface, then a default NUMA node is used for the purpose of selecting a response. Provided that the DAOS Agent can detect any valid network device on any NUMA node, the default response will contain a valid network interface for the client. When a default response is provided, a message in the Agent’s log is emitted:

No network devices bound to client NUMA node X. Using response from NUMA Y

To improve performance, it is worth figuring out if the client bound itself to the wrong NUMA node, or if expected network devices for that NUMA node are missing from the Agent’s fabric scan.

In some situations, the Agent may detect no network devices and the response cache will be empty. In such a situation, the GetAttachInfo response will contain no interface assignment and the following info message will be found in the Agent’s log:

No network devices detected in fabric scan; default AttachInfo response may be incorrect

In either situation, the admin may execute the command daos\_agent net-scan with appropriate debug flags to gain more insight into the configuration problem.

**Disabling the GetAttachInfo cache:**

The default configuration enables the Agent GetAttachInfo cache. If it is desired, the cache may be disabled prior to DAOS Agent startup by setting the Agent’s environment variable DAOS\_AGENT\_DISABLE\_CACHE=true. The cache is loaded only at Agent startup. The following debug message will be found in the Agent’s log:

GetAttachInfo agent caching has been disabled

If the network configuration changes while the Agent is running, it must be restarted to gain visibility to these changes. For additional information, please refer to the [System Deployment: Agent Startup](https://github.com/daos-stack/daos/blob/master/doc/admin/deployment.md#disable-agent-cache-optional) documentation section.