Configuration and execution of MPI parallel I/O API extension using persistent memory

Artur Malinowski and Paweł Czarnul

November 16, 2015

Abstract

This document is a step by step tutorial showing configuration of an environment ready to develop and run MPI parallel I/O API extension using persistent memory.

	Project	**Optimized MPI API for persistent memory**
1	Author	Pawel Czarnul
	Version	0.1
	Modification date	December 16, 2014
	Description	Template
2	Author	Artur Malinowski
	Version	0.2
	Modification date	October 18, 2015
	Description	Main content
3	Author	Pawel Czarnul
	Version	0.3
	Modification date	October 19, 2015
	Description	Slight corrections
4	Author	Artur Malinowski
	Version	0.4
	Modification date	November 15, 2015
	Description	Corrected file path in example

1 Cluster

While it is possible to run the extension on a single node, it is designed to improve performance of cluster environments. A minimum reasonable configuration that should allow to test most of the supported execution scenarios includes three nodes: one node for a distributed file system server, and two nodes for the extension. During development we used eight nodes: seven responsible for computations, one for cluster access and configuration, source compilation, execution of tests and distributed file system server.

The easiest way to automate management of the cluster environment is dedicated software. We used Rocks Cluster 6.1.1. The extension is compatible with POSIX operating systems, so any modern Linux distribution should be able to launch it properly. In the cluster delivered by Intel, CentOS 6.5 was used.

More info:

- Rocks clusters 6.1.1 docs
- CentOS home page

2 DAX + libpmem

It is assumed, that each computing node of the cluster already has two libraries provided by Intel, installed and configured:

1. the Persistent Memory Driver + ext4 Direct Access (DAX);
2. libpmem.

Each node requires a single pmem device with preallocated memory mounted at the same path (e.g. /mnt/pmem). Each node requires read-write permission for pmem device mount point granted for the user who will run the extension. A minimum size of the pmem device is not specified, however the extension allows to read files limited by the total (accumulated) size of pmem devices in a cluster. In our experiments we allocated about 16GB per each computing node that gave maximum supported file size about 100GB.

In case of any problem with `clock_*` functions in linkinkg phase of NVML benchmarks, please check glibc version. If glibc is earlier than 2.17, this patch should be applied.

More info:

- NVML github page (top-level project for libpmem)
- Pmem driver + DAX github page

3 Distributed file system

In MPI IO each process is required to have access to a file that it operates on. The most efficient way to provide access is a distributed file system. We decided to use OrangeFS 2.9.1, because of its performance and optimized MPI-IO support. OrangeFS can be installed without root access from sources using:

```
./configure --prefix=/home/user/orangefs \
        --enable-segv-backtrace \
        --enable-shared
make
make install
```

It should be installed on each cluster node.

3.1 Server side

The whole configuration – created only on a server node – is intuitive and can be set up by an interactive command:

```
/home/user/orangefs/bin/pvfs2-genconfig \
        /home/user/orangefs-config
```

File structures (also only on the node that hosts file system server) are created with:

```
/home/user/orangefs/sbin/pvfs2-server \
        /home/user/orangefs-config -f
```

and the server is started using:

```
/home/user/orangefs/sbin/pvfs2-server \
        /home/user/orangefs-config
```

3.2 Client side

Installation performed without root access does not allow to mount a file system, OrangeFS reads the server location using descriptor located on path specified in `$PVFS2TAB_FILE` variable. A sample client configuration file contains single line:

```
tcp://server_ip:server_port/server_path \
        /home/user/mount_point \
        pvfs2 defaults,noauto 0 0
```

After the aforementioned configuration, files should be accessible using special commands, i.e. `pvfs2-ping`, `pvfs2-cp`, `pvfs2-ls`.

More info:

- OrangeFS quick start guide

4 MPI Implementation

Although there are plenty of MPI implementations, we chose MPICH 3.1.4 because of its compatibility with OrangeFS. Standard compilation from sources should be parameterized with an OrangeFS switch:

```
./configure --with-pvfs2=/home/user/path_to_orangefs_install_dir
make
make install
```

After installation, files stored on OrangeFS should be available using MPI IO when a file name is prefixed by "`pvfs2:`".

More info:

- MPICH Installer's Guide

5 MPI I/O API extension using pmem

Extension installation is as follows:

```
autoreconf -i
./configure CC=/path/to/mpicc
make
make install
```

To run any application from the examples or test directory, mpirun command is required. Example:

```
/path/to/mpirun -machinefile /path/to/machinefile -np 105 \
        mpi_pmem_io_func_test /mnt/pmem /path/to/shared/file
```