DataSpaces Documentation

Philip Davis

May 17, 2024

CONTENTS

DataSpaces is a communication library aimed at supporting interactions between large-scale scientific simulation, analysis, and visualization programs. DataSpaces enables programs to write to and read from shared N-dimensional arrays without centralized query processing or indexing using low-latency RDMA transfers. The result is highly-scalable data access between components of an HPC workflow. DataSpaces can be used to tranfer data in *in-situ* workflows, such as coupled simulations and in-situ analysis workflows, moving data through shared memory and RDMA tranfers, rather than using the file system. Like a shared file system, DataSpaces allow data readers to be decoupled from writers in both space in time. In other words, no sychronization of writers and readers is required, and readers may access data written by any process.

CHAPTER

ONE

CONTENTS

1.1 Installing

The easiest way to install DataSpaces is using Spack. Spack is a package manager aimed at HPC and scientific computing. Using Spack simplifies the installation of DataSpaces and its dependencies.

If you wish to install DataSpaces directly from source, the distribution repo can be found on GitHub.

1.1.1 Installing Spack

To install Spack, follow the getting started instructions found here. This will install the package manager, and make a large variety of packages available.

1.1.2 Installing the DataSpaces repository

The DataSpaces group maintains a repository for the DataSpaces spack package (and any relevant ancillary packages). This can be found here. In order to use this package, you will need to first install Spack using the above instructions. Once you have done this, you can load the DataSpaces package repository by doing the following:

```
git clone https://github.com/rdi2dspaces/dspaces-spack.git
spack repo add dspaces-spack
```

1.1.3 Installing DataSpaces

One the DataSpaces repository has been loaded, the dataspaces package can be installed with:

```
spack install dataspaces
```

This will automatically install allDdataSpaces dependencies and the dataspaces package itself. Once the package has been installed the command:

```
spack load dataspaces
```

Configures the environment to use DataSpaces, adding the server binary's directory to `PATH`, any shared library paths to `LD_LIBRARY_PATH`, etc. This simplifies building and running programs that use DataSpaces.

1.2 How to Use DataSpaces

DataSpaces consists of two components: a client library and a server library. Additionally, the DataSpaces package comes packaged with a standalone, MPI-based server binary. The typical usage of DataSpaces is to run the server binary along-side the user's application, and use the DataSpaces calls provided by the client library to store and access data from the server. It is also possible to run the server in a subset of application proccesses, if it is not desired to run the server as an independent binary.

DataSpaces provides a full set of bindings for C/C++, and a subset of the API for fortran and python. This makes it possible to share data between applications written in different programming languages via the common put/get abstraction.

1.2.1 Building a C/C++ program with DataSpaces

Flags necessary for compiling a program that uses DataSpaces can be found from the pkg-config file installed by DataSpaces in `<INSTALL_ROOT>/lib/pkgconfig`. If installing using spack, the appropriate directory will be added to `PGK_CONFIG_PATH` when the dataspaces module is loaded. `pkg-config` can provide useful information that depends on which flag is provided:

Provides compilation flags for building a program that uses the dataspaces API:

```
pkg-config --cflags dspaces
```

Provides linking flags for building a program that uses the dataspaces API:

```
pkg-config --libs dspaces
```

Provides the path to the dspaces_server binary:

```
pkg-config --variable=exec_prefix dspaces
```

Alternatively, dataspaces installs a CMake targets file that makes it easy to include dspaces in a CMake project. If dataspaces was installed with Spack, `CMAKE_PREFIX_PATH` will be updated when the dataspaces package is loaded. Recent versions of cmake will also be able to find dspaces if `<INSTALL_ROOT>/bin` is in the users `PATH` environment variable.

To include dspaces in a CMake project, simply add `find_package(dspaces)` to the project's CMakeLists.txt file and include `dspaces::dspaces` in the target_link_libraries for whatever target is using dspaces.

1.2.2 Building a Fortran program with DataSpaces

Flags for Fortran compilation cannot be obtained through pkg-config. However, a CMake project can be configured to automatically configure compilation for dataspaces with Fortran. To do this, add `find_package(dspaces)` to the project's CMakeLists.txt file and include `dspaces::fortran` in the target_link_libraries for whatever target is using dspaces.

1.2.3 Using DataSpaces with Python

In order to use the DataSpaces pythong bindings, `<INSTALL_ROOT>/lib/<PYTHONVER>/dist-packages` must be added to `PYTHONPATH`. Spack will do this automatically when the dataspaces package is loaded. To use the Python bindings, import the `dspaces` module.`

1.3 How to Run

1.3.1 Running the Server

The DataSpaces server expects to find the `dataspaces.conf` file in its working directory. The format of this file is a list of configuration values, one per line:

`<variable> = <value>`, e.g. `num_apps = 1`

The possible values are as follows:

num_apps: this value is the number of `dspaces_kill()` calls from clients that are needed to kill the server binary.

ndim: number of dimensions for the default global data domain.

dims: size of each dimension for the default global data domain.

max_versions: maximum number of versions of a data object to be cached in DataSpaces servers.

hash_version: the type of distributed hash table used. A value of `1` means that a Hilbert SFC is partitioned into continuous segments and distributed across the servers. A value of `2` means the space is partitioned by repeating bisection along the longest domain.

NOTES on what values to use:

The global dimensions have implications for performance. Data indexing will be partitioned evently across the global dimensions, and so if data is only being writtent to a subset of the global dimensions there is a risk of unabalanced indexing load. Ideally, the data domain being written to will match the global dimensions as closely as possible. The default value set in `dataspaces.conf` is for convenience. The application can set this per variable with `dspaces_define_gdim()`.

`hash_version = 1` has better locality in the most general case, and should be preferred unless the dimensions of the data domain are not a power of two or the ratio of longest to shortest dimension is greater than two.

`num_apps` should be set in conjunction with how `dspaces_kill()` is used in the application(s) using dataspaces. Generally, one rank of each application should call `dspaces_kill()`, and the number of process groups using dataspaces will be the same as `num_apps`. Occasionally, it is not practical to have a client call `dspaces_kill()`, and the dataspaces repo provides a standalone binary `terminator` to send a single `dspaces_kill()` and then exit.

1.3.2 Bootstrapping communication

The server produces a bootstrap file during its init phase, `conf.ds`. This file must be read by the clients (or rank zero of the clients if `dspaces_init_mpi()` is being used. This file provides the clients with enough information to make initial contact with the server and perform wire-up. In order to find this file, the server and client application must be run in the same working directory, or at last a symlink of `ds.conf` should be present.

1.3.3 Environment variables

There are a few environment variables that can be used to influence DataSpaces.

`DSPACES_DEBUG` - enables substantial debug output for both clients and server.

`DSPACES_DEFAULT_NUM_HANDLERS` - the number of request handling threads launched by the server (in addition to the main thread). Default: 4.

This value should be changed if it is likely to oversubscribe or underutilize the node the server is running on.

The server binary, `dspaces_server`, takes a single argument: the listen_address. This is a Mercury-specific connection string (see Mercury documentation for details.) Common values are: `sockets` to use TCP for communication, `sm` for shared memory (if all clienta and server processes are on the same node) and `ofi+X` for RDMA, where `X` is `verbs`, `psm2`, or `cray` as is appropriate for the system fabric.

1.4 DataSpaces Examples

The DataSpaces repo includes a set of examples. When built using cmake, DataSpaces configures a Makefile and copies these examples in to the *examples/* directory in the build environment. This process can easily be manually duplicated by moving opts.mk.in to opts.mk and filling in a valid value for *CC* (usually *mpicc*).

Each example contains everything needed to build a run a program using dataspaces.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search