# PDC

**Houjun Tang, Qiao Kang, Bin Dong, Quincey Koziol, Suren Byna,**

**Apr 11, 2023**

# GETTING STARTED

Proactive Data Containers (PDC) software provides an object-centric API and a runtime system with a set of data object management services. These services allow placing data in the memory and storage hierarchy, performing data movement asynchronously, and providing scalable metadata operations to find data objects. PDC revolutionizes how data is stored and accessed by using object-centric abstractions to represent data that moves in the high-performance computing (HPC) memory and storage subsystems. PDC manages extensive metadata to describe data objects to find desired data efficiently as well as to store information in the data objects.

PDC API, data types, and developer notes are available in docs/readme.md

More information and publications of PDC is available at https://sdm.lbl.gov/pdc

If you use PDC in your research, please use the following citation:

Byna, Suren, Dong, Bin, Tang, Houjun, Koziol, Quincey, Mu, Jingqing, Soumagne, Jerome, Vishwanath, Venkat, Warren, Richard, and Tessier, François. Proactive Data Containers (PDC) v0.1. Computer Software. https://github.com/hpc-io/pdc. USDOE. 11 May. 2017. Web. doi:10.11578/dc.20210325.1.

CHAPTER

ONE

# GETTING STARTED

Proactive Data Containers (PDC) software provides an object-centric API and a runtime system with a set of data object management services. These services allow placing data in the memory and storage hierarchy, performing data movement asynchronously, and providing scalable metadata operations to find data objects. PDC revolutionizes how data is stored and accessed by using object-centric abstractions to represent data that moves in the high-performance computing (HPC) memory and storage subsystems. PDC manages extensive metadata to describe data objects to find desired data efficiently as well as to store information in the data objects.

PDC API, data types, and developer notes are available in docs/readme.md

More information and publications of PDC is available at https://sdm.lbl.gov/pdc

The following dependencies will need to be installed:

- libfabric
- Mercury

## 1.1 Dependencies

The following instructions are for installing PDC on Linux and Cray machines. GCC version 7 or newer and a version of MPI are needed to install PDC.

Current PDC tests have been verified with MPICH. To install MPICH, follow the documentation in https://www.mpich.org/static/downloads/3.4.1/mpich-3.4.1-installguide.pdf

PDC also depends on libfabric and Mercury. We provide detailed instructions for installing libfabric, Mercury, and PDC below.

**Attention:** Make sure to record the environmental variables (lines that contains the "export" commands). They are needed for running PDC and to use the libraries again.

### 1.1.1 Install libfabric

```
$ wget https://github.com/ofiwg/libfabric/archive/v1.11.2.tar.gz
$ tar xvzf v1.11.2.tar.gz
$ cd libfabric-1.11.2
$ mkdir install
$ export LIBFABRIC_DIR=$(pwd)/install
$ ./autogen.sh
$ ./configure --prefix=$LIBFABRIC_DIR CC=gcc CFLAG="-O2"
$ make -j8
$ make install
$ export LD_LIBRARY_PATH="$LIBFABRIC_DIR/lib:$LD_LIBRARY_PATH"
$ export PATH="$LIBFABRIC_DIR/include:$LIBFABRIC_DIR/lib:$PATH"
```

### 1.1.2 Install Mercury

**Attention:** Make sure the ctest passes. PDC may not work without passing all the tests of Mercury.

Step 2 in the following is not required. It is a stable commit that has been used to test when these these instructions were written. One may skip it to use the current master branch of Mercury.

```
$ git clone https://github.com/mercury-hpc/mercury.git
$ cd mercury
$ git checkout e741051fbe6347087171f33119d57c48cb438438
$ git submodule update --init
$ export MERCURY_DIR=$(pwd)/install
$ mkdir install
$ cd install
$ cmake ../ -DCMAKE_INSTALL_PREFIX=$MERCURY_DIR -DCMAKE_C_COMPILER=gcc -DBUILD_SHARED_
↪LIBS=ON -DBUILD_TESTING=ON -DNA_USE_OFI=ON -DNA_USE_SM=OFF
$ make
$ make install
$ ctest
$ export LD_LIBRARY_PATH="$MERCURY_DIR/lib:$LD_LIBRARY_PATH"
$ export PATH="$MERCURY_DIR/include:$MERCURY_DIR/lib:$PATH"
```

## 1.2 Installation

### 1.2.1 Install PDC

One can replace mpicc to other available MPI compilers. For example, on Cori, cc can be used to replace mpicc. ctest contains both sequential and MPI tests for the PDC settings. These can be used to perform regression tests.

```
$ git clone https://github.com/hpc-io/pdc.git
$ cd pdc
$ git checkout stable
$ cd src
$ mkdir install
```

(continues on next page)

(continued from previous page)

```
$ cd install
$ export PDC_DIR=$(pwd)
$ cmake ../ -DBUILD_MPI_TESTING=ON -DBUILD_SHARED_LIBS=ON -DBUILD_TESTING=ON -DCMAKE_
↪INSTALL_PREFIX=$PDC_DIR -DPDC_ENABLE_MPI=ON -DMERCURY_DIR=$MERCURY_DIR -DCMAKE_C_
↪COMPILER=mpicc
$ make -j8
$ ctest
```

### 1.2.2 Environmental Variables

During installation, we have set some environmental variables. These variables may disappear after the close the current session ends. We recommend adding the following lines to ~/.bashrc. (One may also execute them manually after logging in). The MERCURY_DIR and LIBFABRIC_DIR variables should be identical to the values that were set during the installation of Mercury and libfabric. The install path is the path containing bin and lib directory, instead of the one containing the source code.

```
$ export PDC_DIR="where/you/installed/your/pdc"
$ export MERCURY_DIR="where/you/installed/your/mercury"
$ export LIBFABRIC_DIR="where/you/installed/your/libfabric"
$ export LD_LIBRARY_PATH="$LIBFABRIC_DIR/lib:$MERCURY_DIR/lib:$LD_LIBRARY_PATH"
$ export PATH="$LIBFABRIC_DIR/include:$LIBFABRIC_DIR/lib:$MERCURY_DIR/include:$MERCURY_
↪DIR/lib:$PATH"
```

One can also manage the path with Spack, which is a lot more easier to load and unload these libraries.

## 1.3 Running PDC

The ctest under PDC install folder runs PDC examples using PDC APIs. PDC needs to run at least two applications. The PDC servers need to be started first. The client programs that send I/O request to servers as Mercury RPCs are started next.

We provide a convenient function (mpi_text.sh) to start MPI tests. One needs to change the MPI launching function (mpiexec) with the relevant launcher on a system. On Cori at NERSC, the mpiexec argument needs to be changed to srun. On Theta, it is aprun. On Summit, it is jsrun.

```
$ cd $PDC_DIR/bin
$ ./mpi_test.sh ./pdc_init mpiexec 2 4
```

This is test will start 2 processes for PDC servers. The client program ./pdc_init will start 4 processes. Similarly, one can run any of the client examples in ctest. These source code will provide some knowledge of how to use PDC. For more reference, one may check the documentation folder in this repository.

### 1.3.1 PDC on Cori

Installation on Cori is not very different from a regular linux machine. Simply replacing all gcc/mpicc with the default cc compiler on Cori would work. Add options -DCMAKE_C_FLAGS="-dynamic" to the cmake line of PDC. Add -DCMAKE_C_FLAGS="-dynamic" -DCMAKE_CXX_FLAGS="-dynamic" at the end of the cmake line for mercury as well. Finally, "-DMPI_RUN_CMD=srun" is needed for ctest command later. In some instances and on some systems, unload darshan before installation may be needed.

For job allocation on Cori it is recommended to add "–gres=craynetwork:2" to the command:

```
$ salloc -C haswell -N 4 -t 01:00:00 -q interactive --gres=craynetwork:2
```

And to launch the PDC server and the client, add "–gres=craynetwork:1" before the executables:

Run 4 server processes, each on one node in background:

```
$ srun -N 4 -n  4 -c 2 --mem=25600 --cpu_bind=cores --gres=craynetwork:1 ./bin/pdc_
↪server.exe &
```

Run 64 client processes that concurrently create 1000 objects in total:

```
$ srun -N 4 -n 64 -c 2 --mem=25600 --cpu_bind=cores --gres=craynetwork:1 ./bin/create_
↪obj_scale -r 1000
```

CHAPTER

TWO

# DEFINITIONS

CHAPTER

# THREE

# ASSUMPTIONS

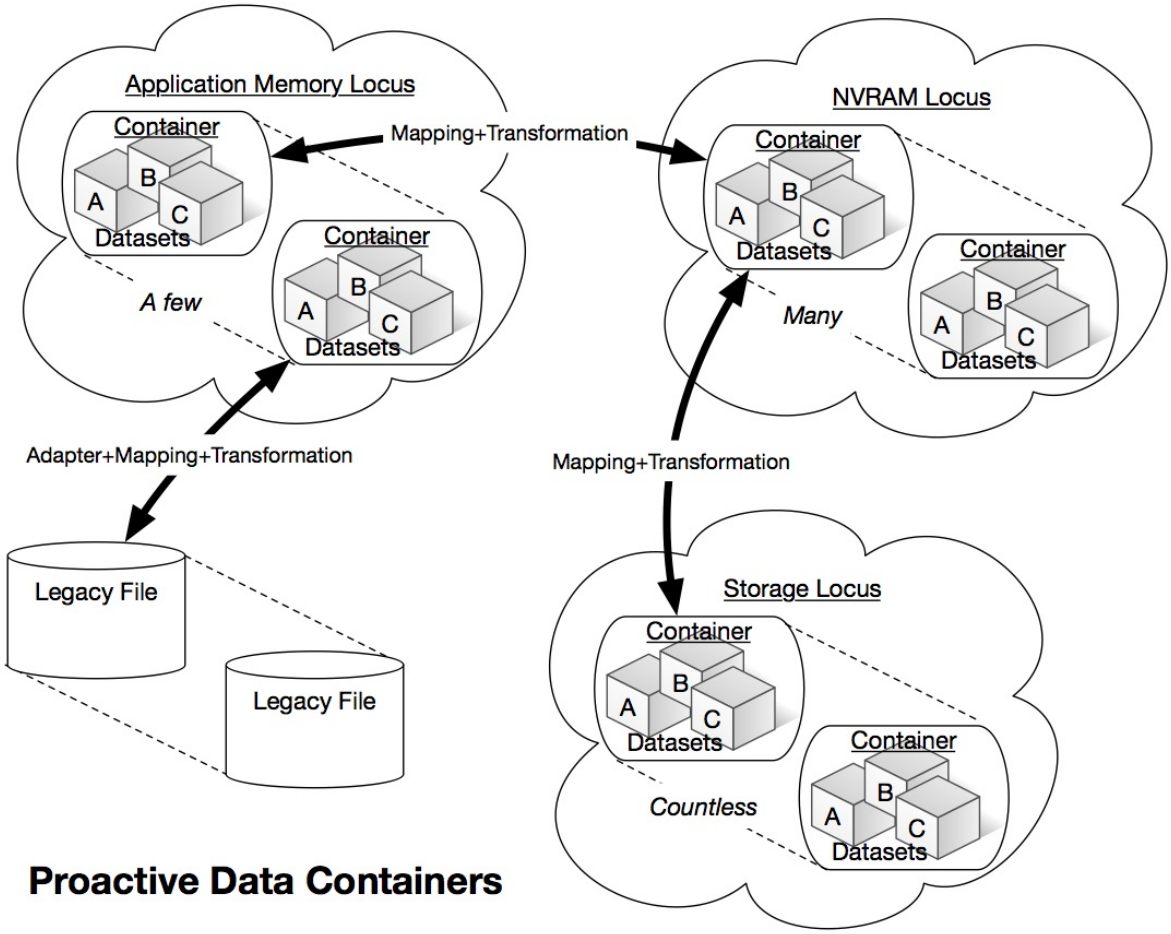
CHAPTER

FOUR

# INTRODUCTION

Emerging high performance computing (HPC) systems are expected to be deployed with an unprecedented level of complexity, due to a very deep system memory/storage hierarchy. This hierarchy is expected to range from CPU cache through several levels of volatile memory to non-volatile memory, traditional hard disks, and tape. Simple and efficient methods of data management and movement through this hierarchy is critical for scientific applications using exascale systems. Existing storage system and I/O (SSIO) technologies face severe challenges in dealing with these requirements. POSIX and MPI I/O standards that are the basis for existing I/O libraries and parallel file systems present fundamental challenges in the areas of scalable metadata operations, semantics-based data movement performance tuning, asynchronous operation, and support for scalable consistency of distributed operations.

Moving toward new paradigms for SSIO in the extreme-scale era, we propose to investigate novel object- based data abstractions and storage mechanisms that take advantage of the deep storage hierarchy and enable proactive automated performance tuning. In order to achieve these overarching goals, we propose a fundamental new data abstraction, called Proactive Data Containers (PDC). A PDC is a container within a locus of storage (memory, NVRAM, disk, etc.) that stores science data in an object-oriented manner. Managing data as objects enables powerful optimization opportunities for data movement and transformations. In this project, we will research: 1) formulation of object-oriented PDCs and their mapping in different levels of the exascale storage hierarchy; 2) efficient strategies for moving data in deep storage hierarchies using PDCs; 3) techniques for transforming and reorganizing data based on application requirements; and 4) novel analysis paradigms for enabling data transformations and user-defined analysis on data in PDCs. The intent of our research is to move the field of HPC SSIO in a direction where it may ultimately be possible to develop scientific applications without the need to perform cumbersome and inefficient tuning to optimize data movement on every system the application runs on.




PDCs will have an impact in many science areas, given the importance of the data management and I/O software stack in achieving science discoveries at scale. The foundations of the novel data management and storage paradigm approaches and formalisms proposed in this research are expected to be applicable to a broad range of scientific and engineering problems that utilize computational and experimental facilities for predictive understanding of physical processes through data analytics and visualization. The proposed techniques are expected to accelerate the crucial process of data-driven exploration and knowledge discovery. While we will work closely with a set of key DOE science applications in the areas of cosmology, climate, genomics, and high-energy density physics to evaluate our research, the proposed new I/O paradigm will be broadly applicable to all users of DOE HPC facilities.

CHAPTER

# FIVE

# HDF5 VOL FOR PDC

CHAPTER

SIX

# PERFORMANCE

CHAPTER

SEVEN

# HELLO PDC EXAMPLE

## 7.1 PDC Hello World

- pdc_init.c
- A PDC program starts with PDCinit and finishes with PDCclose.
- To a simple hello world program for PDC, use the following command.

```
make pdc_init
./run_test.sh ./pdc_init
```

- The script "run_test.sh" starts a server first. Then program "obj_get_data" is executed. Finally, the PDC servers are closed.
- Alternatively, the following command can be used for multile MPI processes.

```
make pdc_init
./mpi_test.sh ./pdc_init mpiexec 2 4
```

- The above command will start a server with 2 processes. Then it will start the application program with 4 processes. Finally, all servers are closed.
- On supercomputers, "mpiexec" can be replaced with "srun", "jsrun" or "aprun".

CHAPTER

EIGHT

# API DOCUMENTATION WITH EXAMPLES

## 8.1 PDC general APIs

- pdcid_t PDCinit(const char *pdc_name)
  - **Input:**
    - pdc_name is the reference for PDC class. Recommended use "pdc"
  - **Output:**
    - PDC class ID used for future reference.
  - All PDC client applications must call PDCinit before using it. This function will setup connections from clients to servers. A valid PDC server must be running.
  - For developers: currently implemented in pdc.c.
- perr_t PDCclose(pdcid_t pdcid)
  - **Input:**
    - PDC class ID returned from PDCinit.
  - **Output:**
    - SUCCEED if no error, otherwise FAIL.
  - This is a proper way to end a client-server connection for PDC. A PDCinit must correspond to one PDCclose.
  - For developers: currently implemented in pdc.c.
- perr_t PDC_Client_close_all_server()
  - **Output:**
    - SUCCEED if no error, otherwise FAIL.
  - Close all PDC servers that running.
  - For developers: see PDC_client_connect.c

## 8.2 PDC container APIs

- **pdcid_t PDCcont_create(const char *cont_name, pdcid_t cont_prop_id)**
  - **Input:**
    - cont_name: the name of container. e.g "c1", "c2"
    - cont_prop_id: property ID for inheriting a PDC property for container.
  - Output: pdc_id for future referencing of this container, returned from PDC servers.
  - Create a PDC container for future use.
  - For developers: currently implemented in pdc_cont.c. This function will send a name to server and receive an container id. This function will allocate necessary memories and initialize properties for a container.
- **pdcid_t PDCcont_create_col(const char *cont_name, pdcid_t cont_prop_id)**
  - **Input:**
    - cont_name: the name to be assigned to a container. e.g "c1", "c2"
    - cont_prop_id: property ID for inheriting a PDC property for container.
  - Output: pdc_id for future referencing.
  - Exactly the same as PDCcont_create, except all processes must call this function collectively. Create a PDC container for future use collectively.
  - For developers: currently implemented in pdc_cont.c.
- **pdcid_t PDCcont_open(const char *cont_name, pdcid_t pdc)**
  - **Input:**
    - cont_name: the name of container used for PDCcont_create.
    - pdc: PDC class ID returned from PDCinit.
  - **Output:**
    - error code. FAIL OR SUCCEED
  - Open a container. Must make sure a container named cont_name is properly created (registered by PDCcont_create at remote servers).
  - For developers: currently implemented in pdc_cont.c. This function will make sure the metadata for a container is returned from servers. For collective operations, rank 0 is going to broadcast this metadata ID to the rest of processes. A struct _pdc_cont_info is created locally for future reference.
- **perr_t PDCcont_close(pdcid_t id)**
  - **Input:**
    - container ID, returned from PDCcont_create.
    - cont_prop_id: property ID for inheriting a PDC property for container.
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Correspond to PDCcont_open. Must be called only once when a container is no longer used in the future.

  - For developers: currently implemented in pdc_cont.c. The reference counter of a container is decremented. When the counter reaches zero, the memory of the container can be freed later.

- **struct pdc_cont_info *PDCcont_get_info(const char *cont_name)**
  - **Input:**
    - name of the container
  - **Output:**
    - Pointer to a new structure that contains the container information See container info (Get Container Info link)
    - Get container information
    - For developers: See pdc_cont.c. Use name to search for pdc_id first by linked list lookup. Make a copy of the metadata to the newly malloced structure.
- **perr_t PDCcont_persist(pdcid_t cont_id)**
  - **Input:**
    - cont_id: container ID, returned from PDCcont_create.
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Make a PDC container persist.
  - For developers, see pdc_cont.c. Set the container life field PDC_PERSIST.
- **perr_t PDCprop_set_cont_lifetime(pdcid_t cont_prop, pdc_lifetime_t cont_lifetime)**
  - **Input:**
    - cont_prop: Container property pdc_id
    - cont_lifetime: See container life time (Get container life time link)
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Set container life time for a property.
  - For developers, see pdc_cont.c.
- **pdcid_t PDCcont_get_id(const char *cont_name, pdcid_t pdc_id)**
  - **Input:**
    - cont_name: Name of the container
    - pdc_id: PDC class ID, returned by PDCinit
  - **Output:**
    - container ID
  - Get container ID by name. This function is similar to open.
  - For developers, see pdc_client_connect.c. It will query the servers for container information and create a container structure locally.
- **perr_t PDCcont_del(pdcid_t cont_id)**
  - **Input:**

        * cont_id: container ID, returned from PDCcont_create.
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Delete a container
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata update.
- **perr_t PDCcont_put_tag(pdcid_t cont_id, char *tag_name, void *tag_value, psize_t value_size)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * tag_name: Name of the tag
        * tag_value: Value to be written under the tag
        * value_size: Number of bytes for the tag_value (tag_size may be more informative)
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Record a tag_value under the name tag_name for the container referenced by cont_id.
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata update.
- **perr_t PDCcont_get_tag(pdcid_t cont_id, char *tag_name, void **tag_value, psize_t *value_size)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * tag_name: Name of the tag
        * value_size: Number of bytes for the tag_value (tag_size may be more informative)
    - **Output:**
        * tag_value: Pointer to the value to be read under the tag
        * error code, SUCCEED or FAIL.
    - Retrieve a tag value to the memory space pointed by the tag_value under the name tag_name for the container referenced by cont_id.
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata retrival.
- **perr_t PDCcont_del_tag(pdcid_t cont_id, char *tag_name)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * tag_name: Name of the tag
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Delete a tag for a container by name
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata update.
- **perr_t PDCcont_put_objids(pdcid_t cont_id, int nobj, pdcid_t *obj_ids)**
    - **Input:**

        * cont_id: Container ID, returned from PDCcont_create.
        * nobj: Number of objects to be written
        * obj_ids: Pointers to the object IDs
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Put an array of objects to a container.
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata update.
- perr_t PDCcont_get_objids(pdcid_t cont_id ATTRIBUTE(unused), int *nobj ATTRIBUTE(unused), pdcid_t **obj_ids ATTRIBUTE(unused) ) TODO:
- **perr_t PDCcont_del_objids(pdcid_t cont_id, int nobj, pdcid_t *obj_ids)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * nobj: Number of objects to be deleted
        * obj_ids: Pointers to the object IDs
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Delete an array of objects to a container.
    - For developers: see pdc_client_connect.c. Need to send RPCs to servers for metadata update.

## 8.3 PDC object APIs

- **pdcid_t PDCobj_create(pdcid_t cont_id, const char *obj_name, pdcid_t obj_prop_id)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * obj_name: Name of objects to be created
        * obj_prop_id: Property ID to be inherited from.
    - **Output:**
        * Local object ID
    - Create a PDC object.
    - For developers: see pdc_obj.c. This process need to send the name of the object to be created to the servers. Then it will receive an object ID. The object structure will inherit attributes from its container and input object properties.
- **PDCobj_create_mpi(pdcid_t cont_id, const char *obj_name, pdcid_t obj_prop_id, int rank_id, MPI_Comm comm)**
    - **Input:**
        * cont_id: Container ID, returned from PDCcont_create.
        * obj_name: Name of objects to be created
        * rank_id: Which rank ID the object is placed to

        * comm: MPI communicator for the rank_id
    - **Output:**
        * Local object ID
    - Create a PDC object at the rank_id in the communicator comm. This function is a colllective operation.
    - For developers: see pdc_mpi.c. If rank_id equals local process rank, then a local object is created. Otherwise we create a global object. The object metadata ID is broadcasted to all processes if a global object is created using MPI_Bcast.

- **pdcid_t PDCobj_open(const char *obj_name, pdcid_t pdc)**
    - **Input:**
        * obj_name: Name of objects to be created
        * pdc: PDC class ID, returned from PDCInit
    - **Output:**
        * Local object ID
    - Open a PDC ID created previously by name.
    - For developers: see pdc_obj.c. Need to communicate with servers for metadata of the object.

- **perr_t PDCobj_close(pdcid_t obj_id)**
    - **Input:**
        * obj_id: Local object ID to be closed.
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Close an object. Must do this after open an object.
    - For developers: see pdc_obj.c. Dereference an object by reducing its reference counter.

- **struct pdc_obj_info *PDCobj_get_info(pdcid_t obj)**
    - **Input:**
        * obj_name: Local object ID
    - **Output:**
      *object information see object information (insert link to object information)
    - Get a pointer to a structure that describes the object metadata.
    - For developers: see pdc_obj.c. Pull out local object metadata by ID.

- **pdcid_t PDCobj_put_data(const char *obj_name, void *data, uint64_t size, pdcid_t cont_id)**
    - **Input:**
        * obj_name: Name of object
        * data: Pointer to data memory
        * size: Size of data
        * cont_id: Container ID of this object
    - **Output:**
        * Local object ID created locally with the input name

  - Write data to an object.
  - For developers: see pdc_client_connect.c. Nedd to send RPCs to servers for this request. (TODO: change return value to perr_t)

- **perr_t PDCobj_get_data(pdcid_t obj_id, void *data, uint64_t size)**
  - **Input:**
    - obj_id: Local object ID
    - size: Size of data
  - **Output:**
    - data: Pointer to data to be filled
    - error code, SUCCEED or FAIL.
  - Read data from an object.
  - For developers: see pdc_client_connect.c. Use PDC_obj_get_info to retrieve name. Then forward name to servers to fulfill requests.
- **perr_t PDCobj_del_data(pdcid_t obj_id)**
  - **Input:**
    - obj_id: Local object ID
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Delete data from an object.
  - For developers: see pdc_client_connect.c. Use PDC_obj_get_info to retrieve name. Then forward name to servers to fulfill requests.
- **perr_t PDCobj_put_tag(pdcid_t obj_id, char *tag_name, void *tag_value, psize_t value_size)**
  - **Input:**
    - obj_id: Local object ID
    - tag_name: Name of the tag to be entered
    - tag_value: Value of the tag
    - value_size: Number of bytes for the tag_value
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Set the tag value for a tag
  - For developers: see pdc_client_connect.c. Need to use PDC_add_kvtag to submit RPCs to the servers for metadata update.
- **perr_t PDCobj_get_tag(pdcid_t obj_id, char *tag_name, void **tag_value, psize_t *value_size)**
  - **Input:**
    - obj_id: Local object ID
    - tag_name: Name of the tag to be entered
  - **Output:**

        * tag_value: Value of the tag
        * value_size: Number of bytes for the tag_value
        * error code, SUCCEED or FAIL.
    - Get the tag value for a tag
    - For developers: see pdc_client_connect.c. Need to use PDC_get_kvtag to submit RPCs to the servers for metadata update.
- **perr_t PDCobj_del_tag(pdcid_t obj_id, char *tag_name)**
    - **Input:**
        * obj_id: Local object ID
        * tag_name: Name of the tag to be entered
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Delete a tag.
    - For developers: see pdc_client_connect.c. Need to use PDCtag_delete to submit RPCs to the servers for metadata update.

## 8.4 PDC region APIs

## 8.5 PDC property APIs

## 8.6 PDC query APIs

- **pdc_query_t *PDCquery_create(pdcid_t obj_id, pdc_query_op_t op, pdc_var_type_t type, void *value)**
    - **Input:**
        * obj_id: local PDC object ID
        * op: one of the followings, see PDC query operators (Insert PDC query operators link)
        * type: one of PDC basic types, see PDC basic types (Insert PDC basic types link)
        * value: constraint value
    - **Output:**
        * a new query structure, see PDC query structure (PDC query structure link)
    - Create a PDC query.
    - For developers, see pdc_query.c. The constraint field of the new query structure is filled with the input arguments. Need to search for the metadata ID using object ID.
- **void PDCquery_free(pdc_query_t *query)**
    - **Input:**
        * query: PDC query from PDCquery_create
    - Free a query structure.

  - For developers, see pdc_client_server_common.c.

- **void PDCquery_free_all(pdc_query_t *root)**
  - **Input:**
    - root: root of queries to be freed
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Free all queries from a root.
  - For developers, see pdc_client_server_common.c. Recursively free left and right branches.
- **pdc_query_t *PDCquery_and(pdc_query_t *q1, pdc_query_t *q2)**
  - **Input:**
    - q1: First query
    - q2: Second query
  - **Output:**
    - A new query after and operator.
  - Perform the and operator on the two PDC queries.
  - For developers, see pdc_query.c
- **pdc_query_t *PDCquery_or(pdc_query_t *q1, pdc_query_t *q2)**
  - **Input:**
    - q1: First query
    - q2: Second query
  - **Output:**
    - A new query after or operator.
  - Perform the or operator on the two PDC queries.
  - For developers, see pdc_query.c
- **perr_t PDCquery_sel_region(pdc_query_t *query, struct pdc_region_info *obj_region)**
  - **Input:**
    - query: Query to select the region
    - obj_region: An object region
  - **Output:**
    - error code, SUCCEED or FAIL.
  - Select a region for a PDC query.
  - For developers, see pdc_query.c. Set the region pointer of the query structure to the obj_region pointer.
- **perr_t PDCquery_get_selection(pdc_query_t *query, pdc_selection_t *sel)**
  - **Input:**
    - query: Query to get the selection
  - **Output:**

        * sel: PDC selection defined as the following. This selection describes the query shape, see PDC selection structure (Insert link to PDC selection structure)
        * error code, SUCCEED or FAIL.
    - Get the selection information of a PDC query.
    - For developers, see pdc_query.c and PDC_send_data_query in pdc_client_connect.c. Copy the selection structure received from servers to the sel pointer.

- **perr_t PDCquery_get_nhits(pdc_query_t *query, uint64_t *n)**
    - **Input:**
        * query: Query to calculate the number of hits
    - **Output:**
        * n: number of hits
        * error code, SUCCEED or FAIL.
    - Get the number of hits for a PDC query
    - For developers, see pdc_query.c and PDC_send_data_query in pdc_client_connect.c. Copy the selection structure received from servers to the sel pointer.

- **perr_t PDCquery_get_data(pdcid_t obj_id, pdc_selection_t *sel, void *obj_data)**
    - **Input:**
        * obj_id: The object for query
        * sel: Selection of the query, query_id is inside it.
    - **Output:**
        * obj_data: Pointer to the data memory filled with query data.
    - Retrieve data from a PDC query for an object.
    - For developers, see pdc_query.c and PDC_Client_get_sel_data in pdc_client_connect.c.

- **perr_t PDCquery_get_histogram(pdcid_t obj_id)**
    - **Input:**
        * obj_id: The object for query
    - **Output:**
        * error code, SUCCEED or FAIL.
    - Retrieve histogram from a query for a PDC object.
    - For developers, see pdc_query.c. This is a local operation that does not really do anything.

- **void PDCselection_free(pdc_selection_t *sel)**
    - **Input:**
        * sel: Pointer to the selection to be freed.
    - **Output:**
        * None
    - Free a selection structure.
    - For developers, see pdc_client_connect.c. Free the coordinates.

- **void PDCquery_print(pdc_query_t *query)**
  - **Input:**
    - query: the query to be printed
  - **Output:**
    - None
  - Print the details of a PDC query structure.
  - For developers, see pdc_client_server_common.c.
- **void PDCselection_print(pdc_selection_t *sel)**
  - **Input:**
    - sel: the PDC selection to be printed
  - **Output:**
    - None
  - Print the details of a PDC selection structure.
  - For developers, see pdc_client_server_common.c.

## 8.7 PDC hist APIs

- **pdc_histogram_t *PDC_gen_hist(pdc_var_type_t dtype, uint64_t n, void *data)**
  - **Input:**
    - dtype: One of the PDC basic types see PDC basic types (Insert link to PDC basic types)
    - n: number of values with the basic types.
    - data: pointer to the data buffer.
  - **Output:**
    - a new PDC histogram structure (Insert link to PDC histogram structure)
  - Generate a PDC histogram from data. This can be used to optimize performance.
  - For developers, see pdc_hist_pkg.c
- **pdc_histogram_t *PDC_dup_hist(pdc_histogram_t *hist)**
  - **Input:**
    - hist: PDC histogram structure (Insert link to PDC histogram structure)
  - **Output:**
    - a copied PDC histogram structure (Insert link to PDC histogram structure)
  - For developers, see pdc_hist_pkg.c
- **pdc_histogram_t *PDC_merge_hist(int n, pdc_histogram_t **hists)**
  - **Input:**
    - hists: an array of PDC histogram structure to be merged (Insert link to PDC histogram structure)
  - **Output:**

        * A merged PDC histogram structure (Insert link to PDC histogram structure)
    - Merge multiple PDC histograms into one
    - For developers, see pdc_hist_pkg.c
- **void PDC_free_hist(pdc_histogram_t *hist)**
    - **Input:**
        * hist: the PDC histogram structure to be freed (Link to Histogram structure)
    - **Output:**
        * None
    - Delete a histogram
    - For developers, see pdc_hist_pkg.c, free structure's internal arrays.
- **void PDC_print_hist(pdc_histogram_t *hist)**
    - **Input:**
        * hist: the PDC histogram structure to be printed (Insert link to histogram structure)
    - **Output:**
        * None
    - Print a PDC histogram's information. The counter for every bin is displayed.
    - For developers, see pdc_hist_pkg.c.

## 8.8 PDC Data types

## 8.9 Basic types

```
typedef enum {
        PDC_UNKNOWN      = -1, /* error                                  */
        PDC_INT          = 0,  /* integer types                          */
        PDC_FLOAT        = 1,  /* floating-point types                   */
        PDC_DOUBLE       = 2,  /* double types                           */
        PDC_CHAR         = 3,  /* character types                        */
        PDC_COMPOUND     = 4,  /* compound types                         */
        PDC_ENUM         = 5,  /* enumeration types                      */
        PDC_ARRAY        = 6,  /* Array types                            */
        PDC_UINT         = 7,  /* unsigned integer types                 */
        PDC_INT64        = 8,  /* 64-bit integer types                   */
        PDC_UINT64       = 9,  /* 64-bit unsigned integer types          */
        PDC_INT16        = 10,
        PDC_INT8         = 11,
        NCLASSES         = 12  /* this must be last                      */
} pdc_var_type_t;
```

## 8.10 Histogram structure

```
typedef struct pdc_histogram_t {
        pdc_var_type_t dtype;
    int            nbin;
    double         incr;
    double        *range;
    uint64_t      *bin;
} pdc_histogram_t;
```

## 8.11 Container info

```
struct pdc_cont_info {
        /*Inherited from property*/
    char                  *name;
    /*Registered using PDC_id_register */
    pdcid_t                local_id;
    /* Need to register at server using function PDC_Client_create_cont_id */
    uint64_t               meta_id;
};
```

## 8.12 Container life time

```
typedef enum {
        PDC_PERSIST,
        PDC_TRANSIENT
} pdc_lifetime_t;
```

## 8.13 Object property public

```
struct pdc_obj_prop *obj_prop_pub {
    /* This ID is the one returned from PDC_id_register . This is a property ID*/
    pdcid_t           obj_prop_id;
    /* object dimensions */
    size_t            ndim;
    uint64_t         *dims;
    pdc_var_type_t    type;
};
```

## 8.14 Object property

```
struct _pdc_obj_prop {
        /* Suffix _pub probably means public attributes to be accessed. */
    struct pdc_obj_prop *obj_prop_pub {
        /* This ID is the one returned from PDC_id_register . This is a property ID*/
        pdcid_t           obj_prop_id;
        /* object dimensions */
        size_t            ndim;
        uint64_t         *dims;
        pdc_var_type_t    type;
    };
    /* This ID is returned from PDC_find_id with an input of ID returned from PDC init.
     * This is true for both object and container.
     * I think it is referencing the global PDC engine through its ID (or name). */
    struct _pdc_class   *pdc{
        char        *name;
        pdcid_t     local_id;
    };
    /* The following are created with NULL values in the PDC_obj_create function. */
    uint32_t            user_id;
    char               *app_name;
    uint32_t            time_step;
    char               *data_loc;
    char               *tags;
    void               *buf;
    pdc_kvtag_t        *kvtag;

    /* The following have been added to support of PDC analysis and transforms.
       Will add meanings to them later, they are not critical. */
    size_t            type_extent;
    uint64_t          locus;
    uint32_t          data_state;
    struct _pdc_transform_state transform_prop{
        _pdc_major_type_t storage_order;
        pdc_var_type_t    dtype;
        size_t            ndim;
        uint64_t          dims[4];
        int               meta_index; /* transform to this state */
    };
};
```

## 8.15 Object info

```
struct pdc_obj_info  {
        /* Directly coped from user argument at object creation. */
    char                   *name;
    /* 0 for location = PDC_OBJ_LOAL.
     * When PDC_OBJ_GLOBAL = 1, use PDC_Client_send_name_recv_id to retrieve ID. */
    pdcid_t                 meta_id;
```

(continues on next page)

(continued from previous page)

```
    /* Registered using PDC_id_register */
    pdcid_t                 local_id;
    /* Set to 0 at creation time. *
    int                     server_id;
    /* Object property. Directly copy from user argument at object creation. */
    struct pdc_obj_prop     *obj_pt;
};
```

## 8.16 Object structure

```
struct _pdc_obj_info {
    /* Public properties */
    struct pdc_obj_info     *obj_info_pub {
        /* Directly copied from user argument at object creation. */
        char                    *name;
        /* 0 for location = PDC_OBJ_LOAL.
        * When PDC_OBJ_GLOBAL = 1, use PDC_Client_send_name_recv_id to retrieve ID. */
        pdcid_t                 meta_id;
        /* Registered using PDC_id_register */
        pdcid_t                 local_id;
        /* Set to 0 at creation time. *
        int                     server_id;
        /* Object property. Directly copy from user argument at object creation. */
        struct pdc_obj_prop     *obj_pt;
    };
    /* Argument passed to obj create*/
    _pdc_obj_location_t     location enum {
        /* Either local or global */
        PDC_OBJ_GLOBAL,
        PDC_OBJ_LOCAL
    }
    /* May be used or not used depending on which creation function called. */
    void                    *metadata;
    /* The container pointer this object sits in. Copied*/
    struct _pdc_cont_info   *cont;
    /* Pointer to object property. Copied*/
    struct _pdc_obj_prop    *obj_pt;
    /* Linked list for region, initialized with NULL at create time.*/
    struct region_map_list *region_list_head {
        pdcid_t                 orig_reg_id;
        pdcid_t                 des_obj_id;
        pdcid_t                 des_reg_id;
        /* Double linked list usage*/
        struct region_map_list *prev;
        struct region_map_list *next;
    };
};
```

## 8.17 Region info

```
struct pdc_region_info {
        pdcid_t                 local_id;
        struct _pdc_obj_info *obj;
        size_t                  ndim;
        uint64_t               *offset;
        uint64_t               *size;
        bool                    mapping;
        int                     registered_op;
        void                   *buf;
};
```

## 8.18 Access type

```
typedef enum { PDC_NA=0, PDC_READ=1, PDC_WRITE=2 }
```

## 8.19 Query operators

```
typedef enum {
    PDC_OP_NONE = 0,
    PDC_GT      = 1,
    PDC_LT      = 2,
    PDC_GTE     = 3,
    PDC_LTE     = 4,
    PDC_EQ      = 5
} pdc_query_op_t;
```

## 8.20 Query structures

```
typedef struct pdc_query_t {
    pdc_query_constraint_t *constraint{
            pdcid_t             obj_id;
            pdc_query_op_t      op;
            pdc_var_type_t      type;
            double              value;   // Use it as a generic 64bit value
            pdc_histogram_t     *hist;

            int                 is_range;
            pdc_query_op_t      op2;
            double              value2;

            void                *storage_region_list_head;
            pdcid_t             origin_server;
            int                 n_sent;
```

(continues on next page)

(continued from previous page)

```
            int                 n_recv;
}
    struct pdc_query_t      *left;
    struct pdc_query_t      *right;
    pdc_query_combine_op_t  combine_op;
    struct pdc_region_info *region;             // used only on client
    void                   *region_constraint;  // used only on server
    pdc_selection_t         *sel;
} pdc_query_t;
```

## 8.21 Selection structure

```
typedef struct pdcquery_selection_t {
pdcid_t  query_id;
size_t   ndim;
uint64_t nhits;
uint64_t *coords;
uint64_t coords_alloc;
} pdc_selection_t;
```

## 8.22 Developers notes

- This note is for developers. It helps developers to understand the code structure of PDC code as fast as possible.
- PDC internal data structure
  - **Linkedlist**
    - Linkedlist is an important data structure for managing PDC IDs.
    - Overall. An PDC instance after PDC_Init() has a global variable pdc_id_list_g. See pdc_interface.h

```
struct PDC_id_type {
PDC_free_t                  free_func;         /* Free function for object's␣
↪of this type    */
PDC_type_t                  type_id;           /* Class ID for the type    ␣
↪                */
        //    const                       PDCID_class_t *cls;/* Pointer to ID␣
↪class                      */
unsigned                    init_count;        /* # of times this type has␣
↪been initialized  */
unsigned                    id_count;          /* Current number of IDs␣
↪held                */
pdcid_t                     nextid;            /* ID to use for the next␣
↪atom               */
DC_LIST_HEAD(_pdc_id_info)  ids;               /* Head of list of IDs      ␣
↪                */
};
```

(continues on next page)

(continued from previous page)

```
struct pdc_id_list {
struct PDC_id_type *PDC_id_type_list_g[PDC_MAX_NUM_TYPES];
};
struct pdc_id_list *pdc_id_list_g;
```

* pdc_id_list_g is an array that stores the head of linked list for each types.
* The _pdc_id_info is defined as the followng in pdc_id_pkg.h.

```
struct _pdc_id_info {
pdcid_t             id;             /* ID for this info                */
hg_atomic_int32_t   count;          /* ref. count for this atom        */
void                *obj_ptr;       /* pointer associated with the atom */
PDC_LIST_ENTRY(_pdc_id_info) entry;
};
```

* obj_ptr is the pointer to the item the ID refers to.
* See pdc_linkedlist.h for implementations of search, insert, remove etc. operations

– **ID**

* ID is important for managing different data structures in PDC.
* e.g Creating objects or containers will return IDs for them

– **pdcid_t PDC_id_register(PDC_type_t type, void *object)**

* This function maintains a linked list. Entries of the linked list is going to be the pointers to the objects. Every time we create an object ID for object using some magics. Then the linked list entry is going to be put to the beginning of the linked list.
* type: One of the followings

```
typedef enum {
        PDC_BADID        = -1,  /* invalid Type                ␣
↪   */
        PDC_CLASS        = 1,   /* type ID for PDC             ␣
↪   */
        PDC_CONT_PROP    = 2,   /* type ID for container property ␣
↪   */
        PDC_OBJ_PROP     = 3,   /* type ID for object property ␣
↪   */
        PDC_CONT         = 4,   /* type ID for container       ␣
↪   */
        PDC_OBJ          = 5,   /* type ID for object          ␣
↪   */
        PDC_REGION       = 6,   /* type ID for region          ␣
↪   */
        PDC_NTYPES       = 7    /* number of library types, MUST BE LAST! ␣
↪   */
} PDC_type_t;
```

* Object: Pointer to the class instance created (bad naming, not necessarily a PDC object).

– **struct _pdc_id_info *PDC_find_id(pdcid_t idid);**

    * Use ID to get struct _pdc_id_info. For most of the times, we want to locate the object pointer inside the structure. This is linear search in the linked list.
    * idid: ID you want to search.

- PDC core classes.
  - **Property**
    * Property in PDC serves as hint and metadata storage purposes.
    * Different types of object has different classes (struct) of properties.
    * See pdc_prop.c, pdc_prop.h and pdc_prop_pkg.h for details.
  - **Container**
    * Container property

```
struct _pdc_cont_prop {
/* This class ID is returned from PDC_find_id with an input of ID returned␣
↪from PDC init. This is true for both object and container.
*I think it is referencing the global PDC engine through its ID (or name).␣
↪*/
        struct _pdc_class *pdc{
        /* PDC class instance name*/
        char        *name;
        /* PDC class instance ID. For most of the times, we only have 1 PDC␣
↪class instance. This is like a global variable everywhere.*/
        pdcid_t     local_id;
};
/* This ID is the one returned from PDC_id_register . This is a property ID␣
↪type.
 * Some kind of hashing algorithm is used to generate it at property create␣
↪time*/
 pdcid_t           cont_prop_id;
/* Not very important */          pdc_lifetime_t    cont_life;
};
```

    * Container structure (pdc_cont_pkg.h and pdc_cont.h)

```
struct _pdc_cont_info {
struct pdc_cont_info    *cont_info_pub {
        /*Inherited from property*/
        char                    *name;
        /*Registered using PDC_id_register */
        pdcid_t                 local_id;
        /* Need to register at server using function PDC_Client_create_cont_
↪id */
        uint64_t                meta_id;
};
/* Pointer to container property.
* This struct is copied at create time.*/
struct _pdc_cont_prop   *cont_pt;
};
```

  - Object
    * Object property See Object Property

* Object structure (pdc_obj_pkg.h and pdc_obj.h) See Object Structure

CHAPTER

NINE

# IN-FLIGHT ANALYSIS

CHAPTER

TEN

# FUTURE WORK

CHAPTER

ELEVEN

# INDICES AND TABLES

- genindex
- modindex
- search