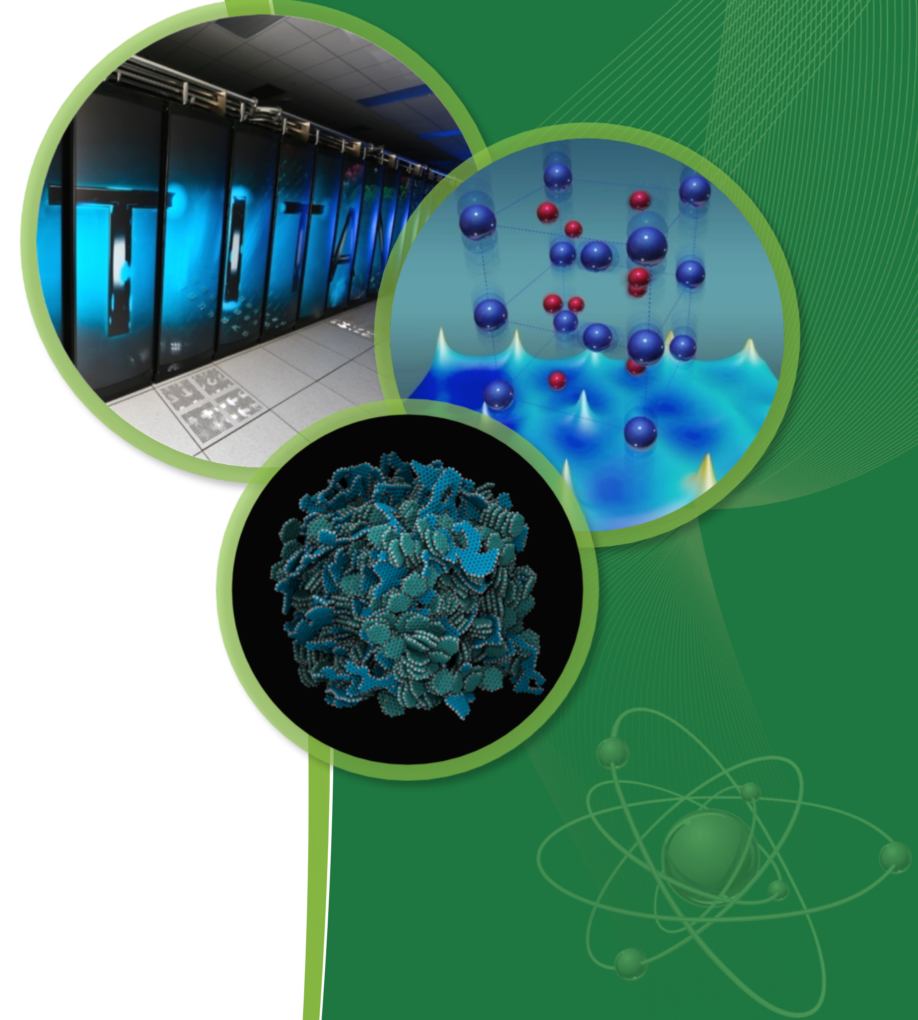
UnifyFS Client Library

API Design Proposal v0.2

January 9, 2020



ORNL is managed by UT-Battelle
for the US Department of Energy

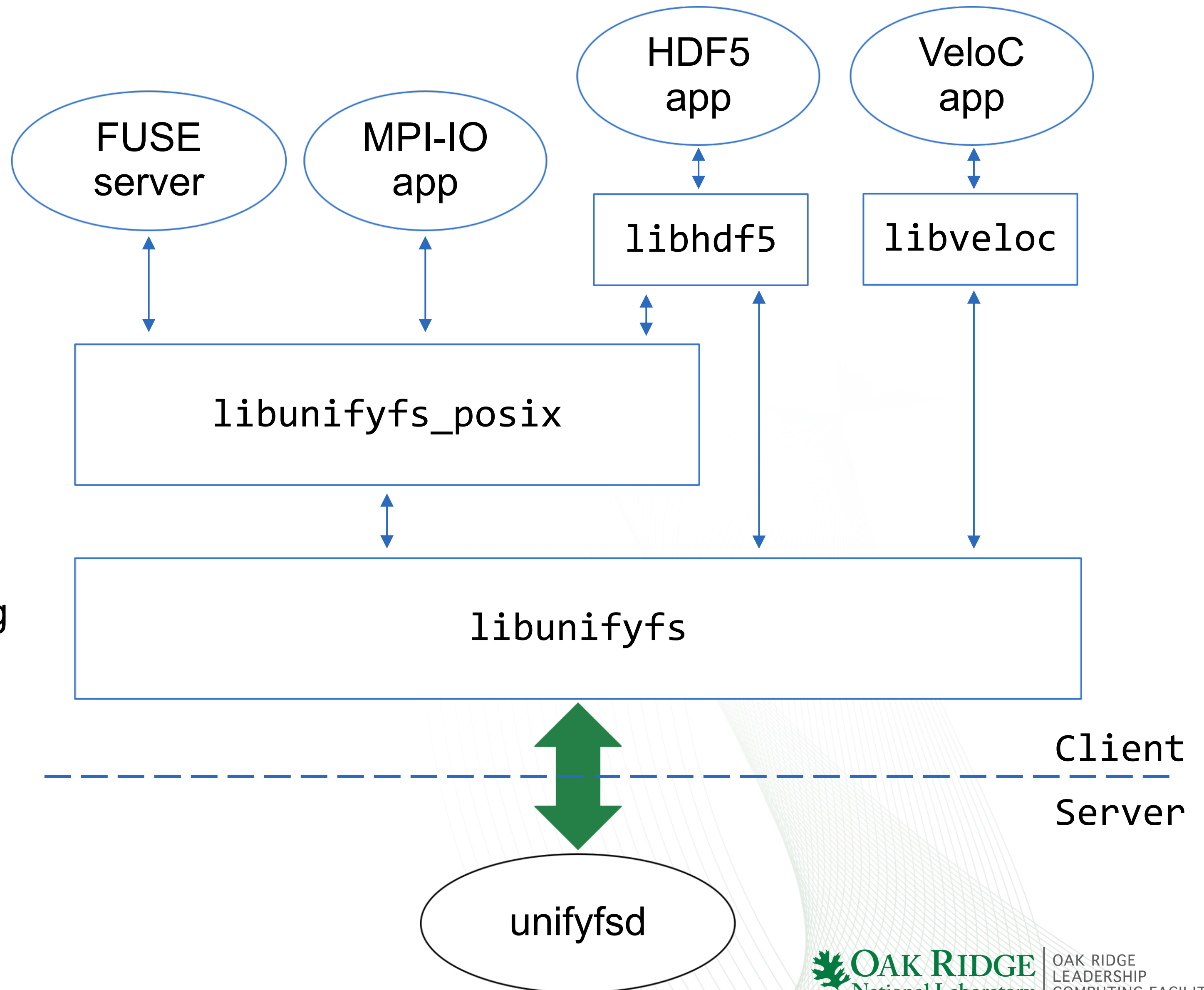
Assumptions

- UnifyFS semantics (i.e., lamination) should be integral to API
- POSIX I/O is not the desired interface
  - though it is the common case
- Asynchronous read/write is fundamental
- local NVM/store provides one (or more) of the following capabilities:
  - regular file system
  - allocatable memory regions

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Library Interactions for Client API

- libunifyfs is the core library
  - handles all server interactions
  - can be used directly by VeloC, HDF5 for explicit control
  - supports custom tools
    - CLI transfer tool
- libunifyfs_posix
  - implements:
    - gfid to local file descriptor mapping
    - POSIX wrappers
  - used by clients that expect POSIX
    - MPI-IO
    - FUSE server
    - unifyfs-shell



OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

UnifyFS Lamination Semantics

- Distributed shared files represented as collection of non-overlapping local file extents
  - one client should `create()`, rest should `open()`
  - one client should `laminate()`

This matches existing application behavior when using parallel file systems

- Before lamination:
  - local writes OK
  - local reads OK
    - Q: is it OK for data written by another client on same node? relatively easy to support, but current semantics say no
- After lamination:
  - no writes
  - global reads OK

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Library initialization and finalization

```
typedef struct unifyfs_client_handle* unifyfs_handle;
```

- fs handle is pointer to opaque struct

```
unifyfs_rc unifyfs_initialize(const unifyfs_options* opts,
unifyfs_handle* fshdl);
```

- `opts`: options to control runtime behavior (see next slide)
- on success, sets `fshdl`

```
unifyfs_rc unifyfs_finalize(unifyfs_handle fshdl);
```


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Library init/fini – FS options

```
typedef struct unifyfs_options {
        /* namespace prefix (default: '/unifyfs') */
        char* fs_prefix;

        /* file system persist path (default: NULL) */
        char* persist_path;

        /* auto-laminate files opened for writing at close? (default: 0) */
        int laminate_at_close;

        /* auto-transfer laminated files to persist path upon fini (default: 0) */
        int transfer_at_finalize;

        /* application debug rank (default: 0) */
        int debug_rank;
} unifyfs_options;
```

Client APIs – Shared Files

- Global file id (gfid)
  - `unifyfs_gfid gfid;`
  - same value across all clients (hash on file path)
  - any translation to process-specific id (e.g., POSIX fd) should be handled separately

- Supports named objects or memory regions
  - give unique (dummy) path for name
  - use transfer API to persist to file system

- Directories are a side-effect of shared file paths
  - no explicit directory create/delete
  - still need to maintain tree/index and support navigation

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Path-oriented Operations

```
unifyfs_rc unifyfs_create(unifyfs_handle fshdl, const int flags, const char* path,
unifyfs_gfid* gfid);
```

- `flags`: should these be same as `creat()`?
- `path`: includes prefix?

```
unifyfs_rc unifyfs_open(unifyfs_handle fshdl, const int flags, const char* path,
unifyfs_gfid* gfid);
```

- `flags`: should these be same as `open()`?
- `path`: includes prefix?

```
unifyfs_rc unifyfs_laminate(unifyfs_handle fshdl, const char* path);
```

- synchronous global lamination (i.e., after this returns, no client should be able to successfully write or open for writing)
- should fail if there are outstanding writes to file that have not been locally

```
unifyfs_rc unifyfs_remove(unifyfs_handle fshdl, const char* filepath);
```


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Metadata Operations

```
unifyfs_rc unifyfs_close(unifyfs_handle fshdl, unifyfs_gfid gfid);
```

- no implied laminate

```
unifyfs_rc unifyfs_laminate_local(unifyfs_handle fshdl, unifyfs_gfid gfid);
```

- client-local asynchronous laminate – “I’m done writing”

```
unifyfs_rc unifyfs_stat(unifyfs_handle fshdl, unifyfs_gfid gfid, unifyfs_status* st);
```

- similar to `fstat()`
- Q: local information only until laminated?
- Q: do we want more targeted methods for retrieving local/global file size and local bytes written?
- Q: do we want a path based `unifyfs_stat()` as well?

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Asynchronous Data Operations

- Types
  - `enum {NOP, READ, WRITE, TRUNCATE, ZERO}` **`unifyfs_ioreq_op`**`;`
  - `enum {INVALID, IN_PROGRESS, CANCELED, COMPLETED}` **`unifyfs_ioreq_state`**`;`
  - `struct { int error; int rc; size_t count; }` **`unifyfs_ioreq_result`**`;`
  - `int (*`**`unifyfs_req_notify_fn`**`)(unifyfs_io_request* req, void *user_data);`
    - asynchronous request completion function
    - may be called by different thread than original requestor

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Asynchronous Data Operations

- I/O request structure

```
typedef struct unifyfs_io_request {
        unifyfs_ioreq_op op;
        unifyfs_gfid gfid;
        off_t offset;
        size_t nbytes;
        void* buf;
        unifyfs_ioreq_state req_state;
        unifyfs_ioreq_result req_result;
        unifyfs_req_notify_fn fn;
        void* notify_user_data;
        int _reqid;
 } unifyfs_io_request;
```


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Asynchronous Data Operations

```
unifyfs_rc unifyfs_dispatch_io(unifyfs_handle fshdl, size_t n_reqs,
unifyfs_io_request* reqs);
```

- dispatch a batch of asynchronous I/O requests

```
unifyfs_rc unifyfs_cancel_io(unifyfs_handle fshdl, size_t n_reqs,
unifyfs_io_request* reqs);
```

- cancel a batch of asynchronous I/O requests

```
unifyfs_rc unifyfs_wait_io(unifyfs_handle fshdl, size_t n_reqs, unifyfs_io_request*
reqs, int waitall=0);
```

- wait on a batch of asynchronous I/O requests
- `waitall`: if non-zero, block until all `reqs` completed or canceled (else, return when any request ready)
- can reuse `reqs` array after subset completed (just update completed `req->op` to `NOP`)

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Asynchronous File Transfer Operations

- File transfer request structure

```
typedef struct unifyfs_transfer_request {
     unifyfs_transfer_mode mode; // transfer mode (e.g., COPY, MOVE, RAW)
     const char* src_path;
     const char* dst_path;
     unifyfs_ioreq_state req_state;
     unifyfs_ioreq_result req_result;
     unifyfs_ioreq_notify_fn fn;
     void* notify_user_data;
     int _reqid;
 } unifyfs_transfer_request;
```

  – one or both of `src/dst_path` should contain prefix

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Asynchronous File Transfer Operations

```
unifyfs_rc unifyfs_dispatch_transfer(unifyfs_handle fshdl, size_t n_reqs,
unifyfs_transfer_request* reqs);
```

- dispatch a batch of asynchronous transfer requests

```
unifyfs_rc unifyfs_wait_transfer(unifyfs_handle fshdl, size_t n_reqs,
unifyfs_transfer_request* reqs, int waitall=0);
```

- wait on a batch of asynchronous transfer requests
- `waitall`: if non-zero, block until all `reqs` completed (else, return when any request ready)

```
unifyfs_rc unifyfs_cancel_transfer(unifyfs_handle fshdl, const size_t nreqs,
unifyfs_transfer_request* reqs);
```


OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Memory-mapped Operations

```
void* addr = unifyfs_map(unifyfs_handle fshdl, unifyfs_gfid gfid,
off_t offset, size_t size);
```

- similar to `mmap()`

```
unifyfs_rc unifyfs_unmap(unifyfs_handle fshdl, unifyfs_gfid gfid,
void* addr);
```

- similar to `munmap()`

```
unifyfs_rc unifyfs_map_sync(unifyfs_handle fshdl, unifyfs_gfid gfid,
void* addr);
```

- similar to `msync()`

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY

Client APIs – Directory Operations

```
unifyfs_rc unifyfs_dir_list(unifyfs_handle fshdl, int flags, const char* dir_path,
                  /* OUT */ size_t* n_ent, const char*** entries);
```

- list directory entries for `dir_path`
- `entries` array is allocated by library, freed by client
- `flags`: LIST_FILES, LIST_SUBDIRS, LIST_ALL

```
unifyfs_rc unifyfs_dir_prune(unifyfs_handle fshdl, const char* dir_path);
```

- recursively remove any empty directories under `dir_path`
- NOTE: an empty directory results from using `unifyfs_remove()` on all its files

Q: **`unifyfs_dir_walk`**`(fshdl, op, path)` instead of **`unifyfs_dir_prune`**`()`, with `PRUNE` as an op type?

- what would the other walk op types be? `REMOVE, COPY_TRANSFER, MOVE_TRANSFER, LAMINATE`

OAK RIDGE National Laboratory | OAK RIDGE LEADERSHIP COMPUTING FACILITY