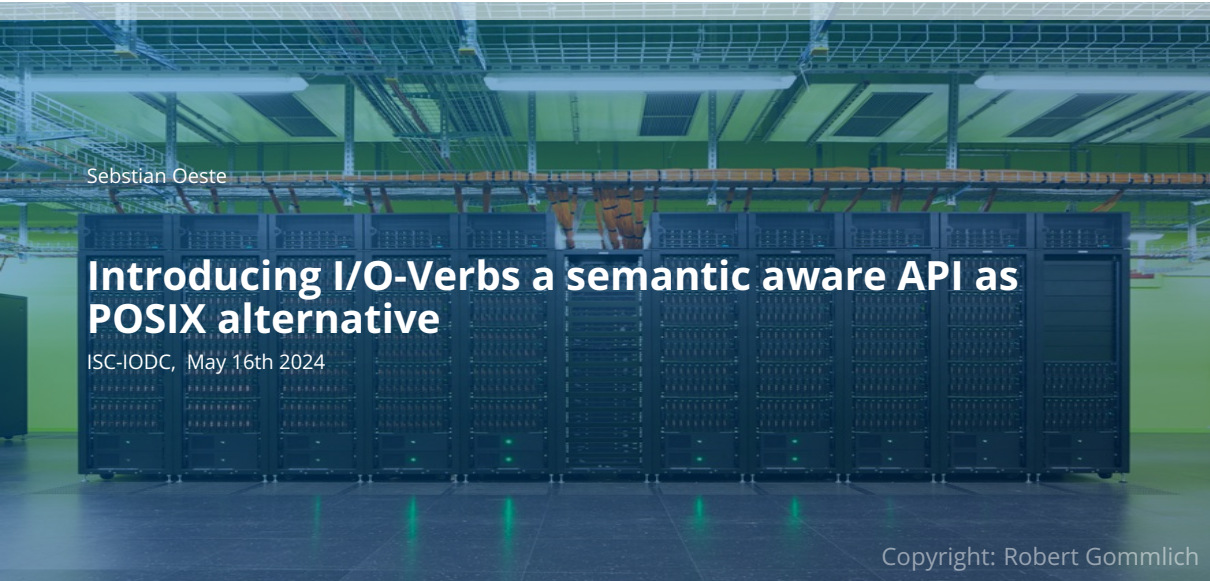
TECHNISCHE UNIVERSITÄT DRESDEN

DRESDEN concept

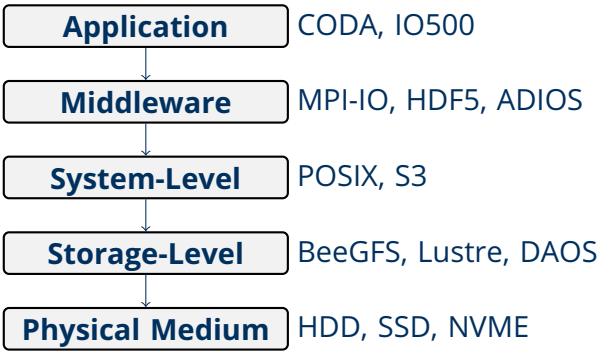


Sebstian Oeste

Introducing I/O-Verbs a semantic aware API as POSIX alternative

ISC-IODC, May 16th 2024

Current I/O Stack



Analysis of I/O API Semantics

Interface	Concurrency Access	Persistency	Consistency	Spatiality	Temporality	Mutability
MPIIO	✔	━	sequential/session	✔	✔	✔
HDF5	✔	━	sequential	✔	✔	✔
Zarr	━	━	sequential	━	✔	✔
POSIX I/O	✖	━	sequential	━	✔	✔
S3	✖	━	sequential	━	✖	━
NFS	✖	━	session	✖	✖	✔
DAOS	✖	━	commit	✔	✖	✔
Lustre	✖	━	sequential	✔	✔	✔
GekkoFS	✖	━	sequential/eventual	━	━	✔
NVME SSD	✖	✔	eventual	━	✖	✖

Table: Comparision of the semantics of IO interfaces. ✔indicates the transport of this semantic category is explicitly supported by the interface. ━ semantics can be derived via indirect support from the interface (e.g. paths). ✖indicates that there is no support for this semantic category in the interface.

POSIX :,(

- Old, well known story … POSIX I/O
  - No notion for parallelism
  - Ridgid consistency semantics
  - Prescriptive metadata
  - Stateful file descriptors
  - API cannot express the intention of the user
- Restrictive behaviour is not necessary for most applications [1][2]

`https://www.nextplatform.com/2017/09/11/whats-bad-posix-io/`

What has been done on that?

- Middleware libraries trying to translate user semantics to lower level libraries e.g ADIOS [3], HDF5 [4], …
- Storage systems with specialized or relaxed semantics e.g. object stores, DAOS [5], GekkoFS [6]
- Additional systems monitoring I/O and infer a optimized configuration for the storage system - e.g. Labios [7], Mirmir [8]

Proposed IO-Stack

Application	CODA, IO500
Middleware	MPI-IO, HDF5, Zarr, ADIOS
System-Level	**Semantic aware system level API**
Storage-Level	BeeGFS, Lustre, (p)NFS, DAOS
Physical Medium	HDD, SSD, NVME

I/O-Verbs - Idea

- Describe I/O on the low level and enabling parallel I/O
- Transport access semantics down to the storage system
- Extensible metadata via key-value pairs
- gather operations in batches with transactional error semantics
- flat metadata namespace
- no hierachical directory structure
- Allows high-level libraries to be layered on top
- asynchronous API
- C and C++ API
- Agnostic of the storage system, able to utilize different backends.
  - Integration into BeeGFS UserSpace Client
  - Fraunhofer IML (In-Memory-Solution)

Architecture

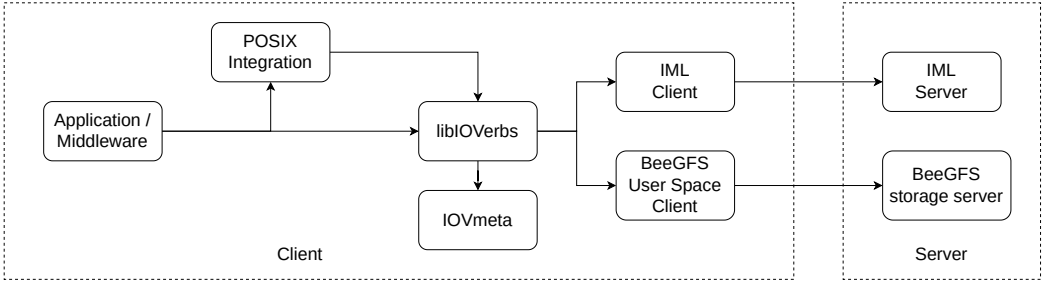


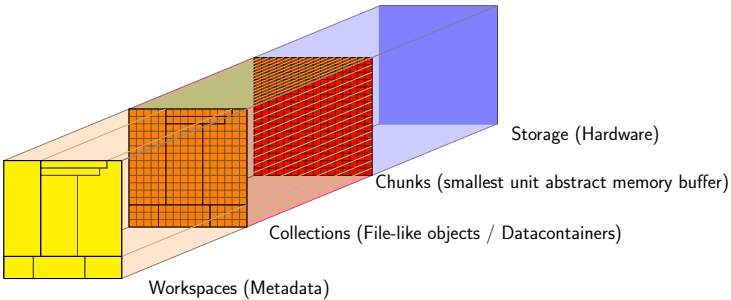
Figure: Architecture of the I/O-Verbs ecosystem.

Transport Semantics

- Semantic Controls - act like a contract between application and storage backend
- describe the semantics of the access on a ressource
- semantic controls will not be persistent
  - Concurency: describe the parallel access on ressource
  - Consistency: when is an update visible for subsequent operations
  - Persistency: when is an update persitent on durable storage
  - Mutability: if and how will the ressource mutated
  - Error: how to proceed if an error occurs (just for batches)
- Intents (hints) provide additional information on the access to a resource:
  - Temporal and spatial Pattern

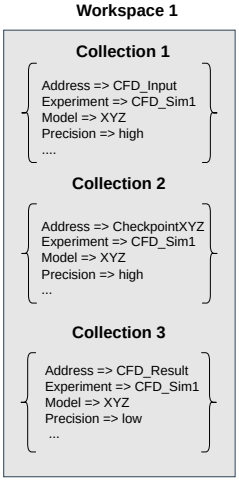
Structural Components

- Workspace - abstraction of a flat metadata namespace
- Collection - file like data containers
- Chunk - memory buffers



Workspace - metadata namespace

- encapsulates a seperate namespace
- needs to be allocated/loaded before use
- holds metadata for collections associated with this workspace
- unique identifier to be adressed
- in POSIX-World comparable with a directory / mountpoint
- controls define required metadata semantics



Workspace API

```
// create a new workspace
auto create_workspace(meta::UniqueName, meta::MetaData,
    Concurrency_Ctl,
    Consistency_Ctl,
    Mutability_Ctl) -> Request<MetaDataCreateRequestData>;

// load an existing workspace
auto load_workspace(meta::UniqueName,
    Concurrency_Ctl,
    Consistency_Ctl,
    Mutability_Ctl) -> Request<MetaDataCreateRequestData>;

// search for existing workspaces
auto lookup_workspace(meta::MetaData) -> Request<MetaDataLookupRequestData>;

// delete existing workspaces
auto delete_workspace(meta::MetaData) -> Request<MetaDataDeleteRequestData>;
```

Collection

Collection - data containers

- can be addressed within the workspace
- contains a vector of chunks
- in POSIX-World comparable with a file
- controls define required data semantics

Collection API

```
// create a new collection
auto create_collection(WorkspaceHandle, meta::UniqueName, meta::MetaData,
    Concurrency_Ctl, Consistency_Ctl,
    Temporality_Ctl,
    Mutability_Ctl) -> Request<MetaDataCreateRequestData>;

// load an existing collection
auto load_collection(WorkspaceHandle, meta::UniqueName,
    Concurrency_Ctl, Consistency_Ctl,
    Temporality_Ctl,
    Mutability_Ctl) -> Request<MetaDataCreateRequestData>;

// search for existing collection
auto lookup_collection(WorkspaceHandle, meta::MetaData) ->
    ↪ Request<MetaDataLookupRequestData>;

// delete existing collection
auto delete_collection(WorkspaceHandle, meta::MetaData) ->
    ↪ Request<MetaDataDeleteRequestData>;
```

Chunk

Chunk - abstraction of a memory buffer

- a chunk is memory buffer for the actual data
- consists of an address, size and offset
- no additional metadata on chunk-level
- opaque to user
- managed by the storage backend

Metadata

- Representation as key-value pairs
- Unique name for addressing
- abritary number of key value pairs possible
- search through kv-space for scientific metatdata
- storage backend defines minimum subset

Data access and modification

- indepentent read/write functions
- always explicit offsets
- batch - gather I/O operations, execute together
- error control - define if execution should proceed or stop on error

Data modification API

```
// indepentent write function
template<typename T, auto SIZE>
auto write(CollectionHandle,
        std::span<T, SIZE>,
        off_t) -> Request<WriteRequestData>;

// indepentent read function
template<typename T, auto SIZE>
auto read(CollectionHandle,
       std::span<T, SIZE>,
       off_t) -> Request<ReadRequestData>;
```

IO-Verbs Batch example

```
// fill the batch operations in a vector
std::vector<iov::Request<iov::MetadataRequestData>> requests;
for (auto idx = 0; idx < mdtest_easy_write_counter; ++idx) {
    std::string collection_name = "file " + mpi_rank + "-" + idx;
    std::string fake_structure = "dir" + mpi_rank;
    requests.push_back(
            iov::create_collection(ws,
                iov::meta::Identifier { {"name", collection_name}, {"subdir",
                    ↪ fake_structure} },
                concurrency_ctl, consistency_ctl,
                temporality_ctl, mutability_ctl));
}
// create batch with vector of md requests and error ctl
auto batch = iov::create_batch(requests, error_ctl);
// do whatever you what ...
for (auto& res : batch.wait_all()) { // block until results are there
    // get somehow the collection handle out of results
}
```

Partner

- Funded by BMBF
- Cooperation of
  - Uni Göttingen
  - GWDG
  - TU Dresden
  - Fraunhofer ITWM
  - DLR
  - ThinkParQ

Discussion

- **Thank you**
- Find I/O-Verbs on GitHub (initial check-in in Sept. 24):
  `https://github.com/IOVerbs`

References I

[1] Chen Wang, Kathryn Mohror, and Marc Snir. “File System Semantics Requirements of HPC Applications”. In: *Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing*. HPDC ’21. Virtual Event, Sweden: Association for Computing Machinery, 2021, pp. 19–30. ISBN: 9781450382175. DOI: 10.1145/3431379.3460637. URL: https://doi.org/10.1145/3431379.3460637.

[2] Sebastian Oeste et al. “Analyzing Parallel Applications for Unnecessary I/O Semantics that Inhibit File System Performance”. In: *High Performance Computing*. Ed. by Amanda Bienz et al. Cham: Springer Nature Switzerland, 2023, pp. 161–176. ISBN: 978-3-031-40843-4.

References II

[3] Jay Lofstead et al. “Adaptable, Metadata Rich IO Methods for Portable High Performance IO”. In: *2009 IEEE International Symposium on Parallel & Distributed Processing*. Rome, Italy: IEEE, May 2009, pp. 1–10. ISBN: 978-1-4244-3751-1. DOI: 10.1109/IPDPS.2009.5161052. (Visited on 06/14/2023).

[4] Jialin Liu et al. “Evaluation of HPC Application I/O on Object Storage Systems”. In: *2018 IEEE/ACM 3rd International Workshop on Parallel Data Storage & Data Intensive Scalable Computing Systems (PDSW-DISCS)*. 2018, pp. 24–34. DOI: 10.1109/PDSW-DISCS.2018.00005.

References III

[5] Zhen Liang et al. “DAOS: A Scale-Out High Performance Storage Stack for Storage Class Memory”. In: *Supercomputing Frontiers*. Ed. by Dhabaleswar K. Panda. Cham: Springer International Publishing, 2020, pp. 40–54. ISBN: 978-3-030-48842-0.

[6] Marc-André Vef et al. “GekkoFS—A temporary burst buffer file system for HPC applications”. In: *Journal of Computer Science and Technology* 35 (2020), pp. 72–91.

[7] Anthony Kougkas, Hariharan Devarajan, and Xian-He Sun. “Bridging Storage Semantics Using Data Labels and Asynchronous I/O”. In: *ACM Trans. Storage* 16.4 (Oct. 2020). ISSN: 1553-3077. DOI: 10.1145/3415579. URL: https://doi.org/10.1145/3415579.

References IV

[8] Hariharan Devarajan and Kathryn Mohror. “Mimir: Extending I/O Interfaces to Express User Intent for Complex Workloads in HPC”. In: *2023 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2023, pp. 178–188. DOI: 10.1109/IPDPS54959.2023.00027.