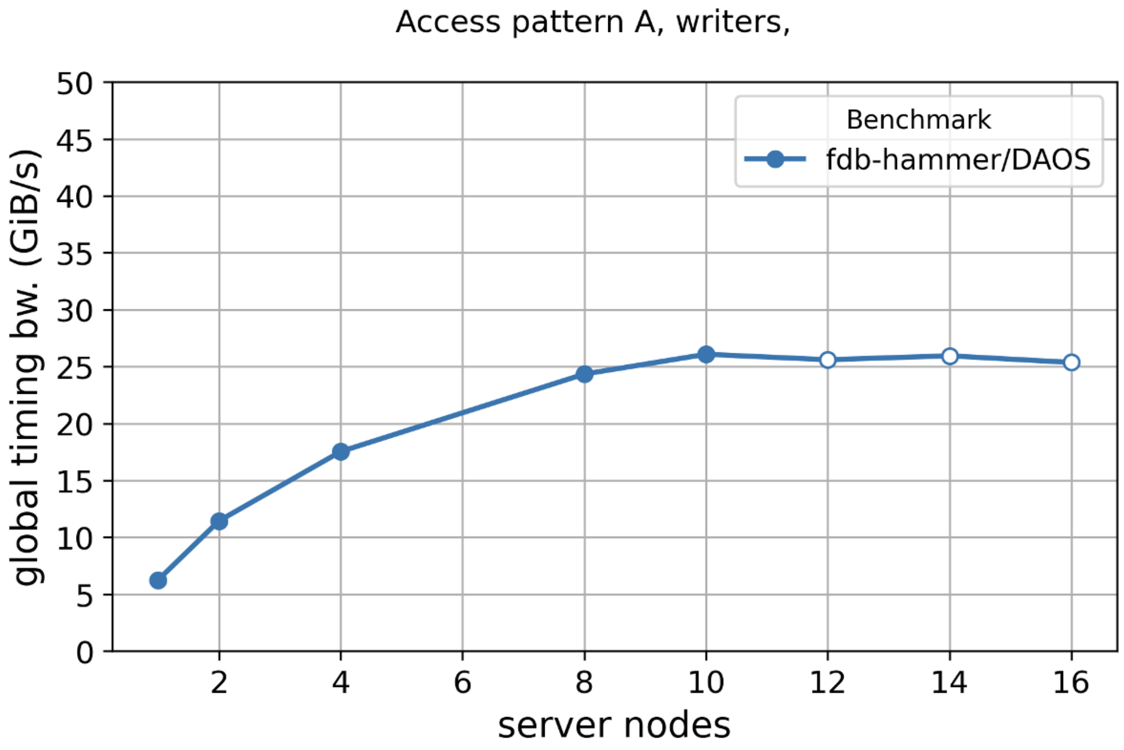
SUMMARY

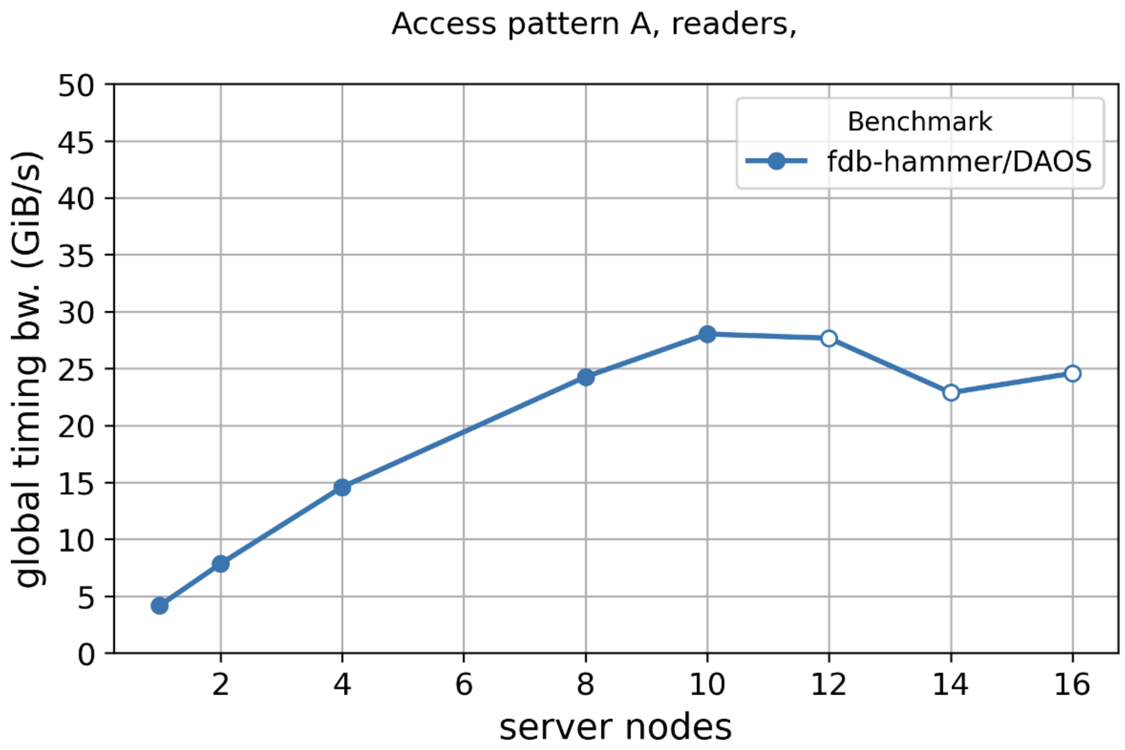
Aside: DAOS Fortran interfacing

```
type, public, bind(c) :: daos_array_stbuf_t
     integer (kind=daos_size_t) :: st_size
     integer (kind=daos_epoch_t) :: st_max_epoch
  end type daos_array_stbuf_t


interface


   integer(kind=c_int) function daos_array_create(coh, oid, th, cell_size, chunk_size,
oh, ev) bind(c,name="daos_array_create")
     import :: c_int
     import :: daos_handle_t
     import :: daos_obj_id_t
     import :: daos_size_t
     import :: daos_event_t
     type(daos_handle_t), value, intent(in) :: coh
     type(daos_obj_id_t), value, intent(in) :: oid
     type(daos_handle_t), value, intent(in) :: th
     integer(kind=daos_size_t), value, intent(in) ::cell_size
     integer(kind=daos_size_t), value, intent(in) ::chunk_size
     type(daos_handle_t), intent(inout) :: oh
     type(daos_event_t), intent(inout) :: ev
   end function daos_array_create
```

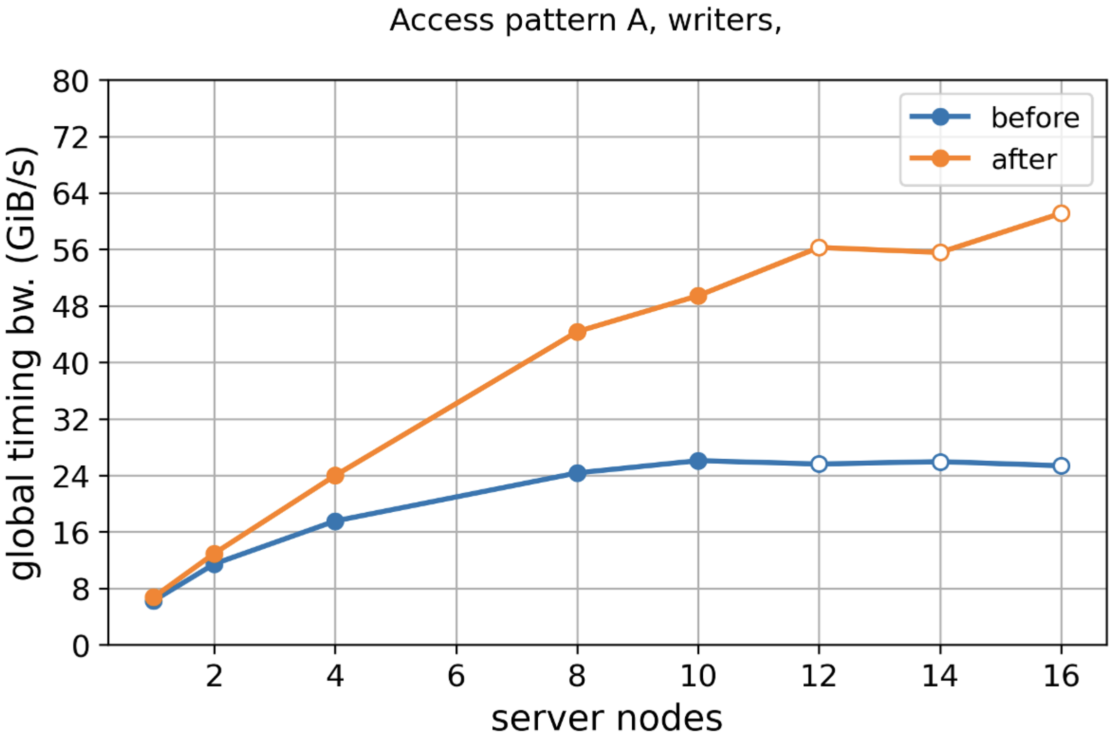
intel ECMWF Google Cloud epcc

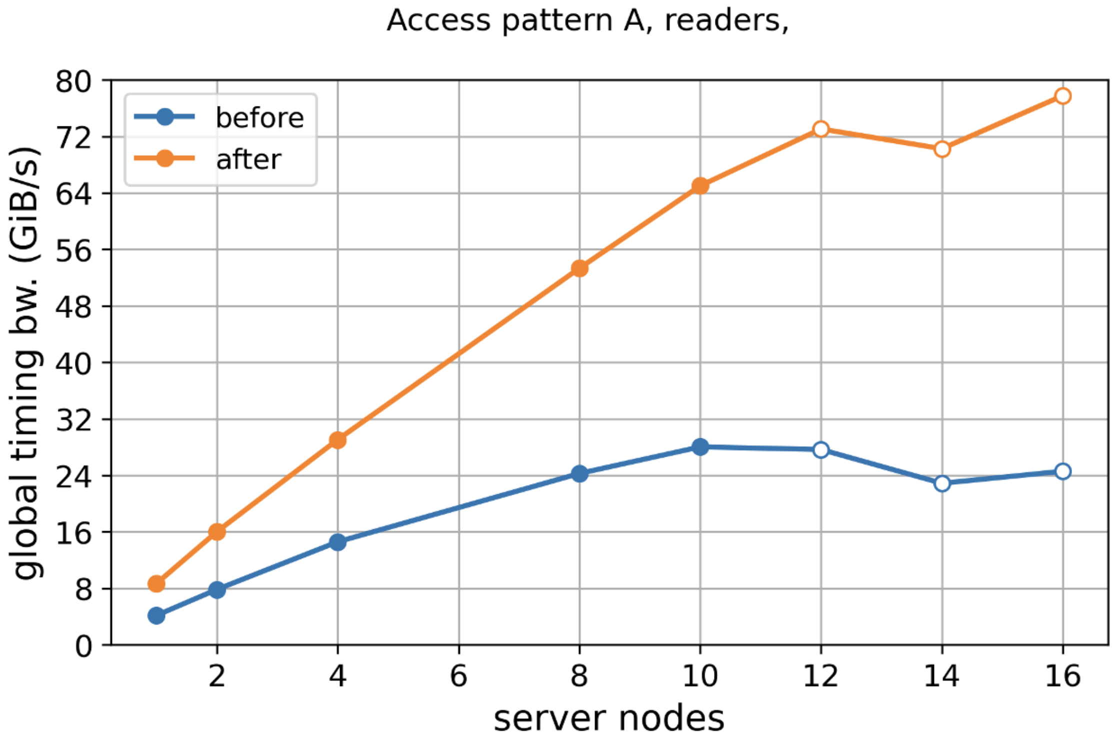
Evaluate performance/approach





Optimised performance

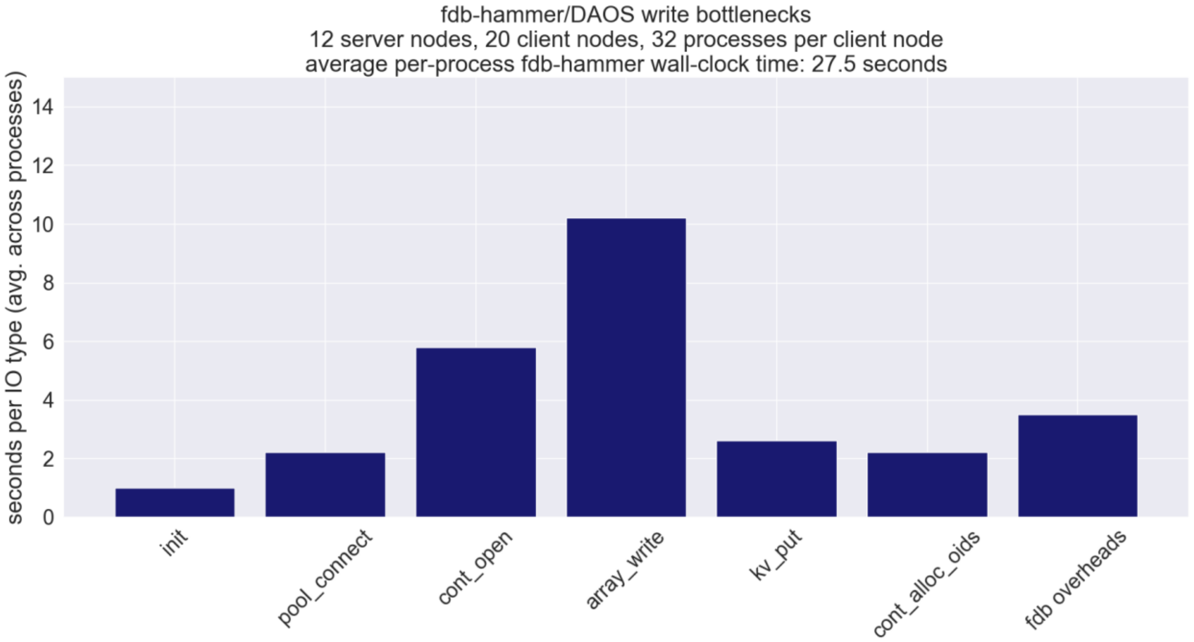


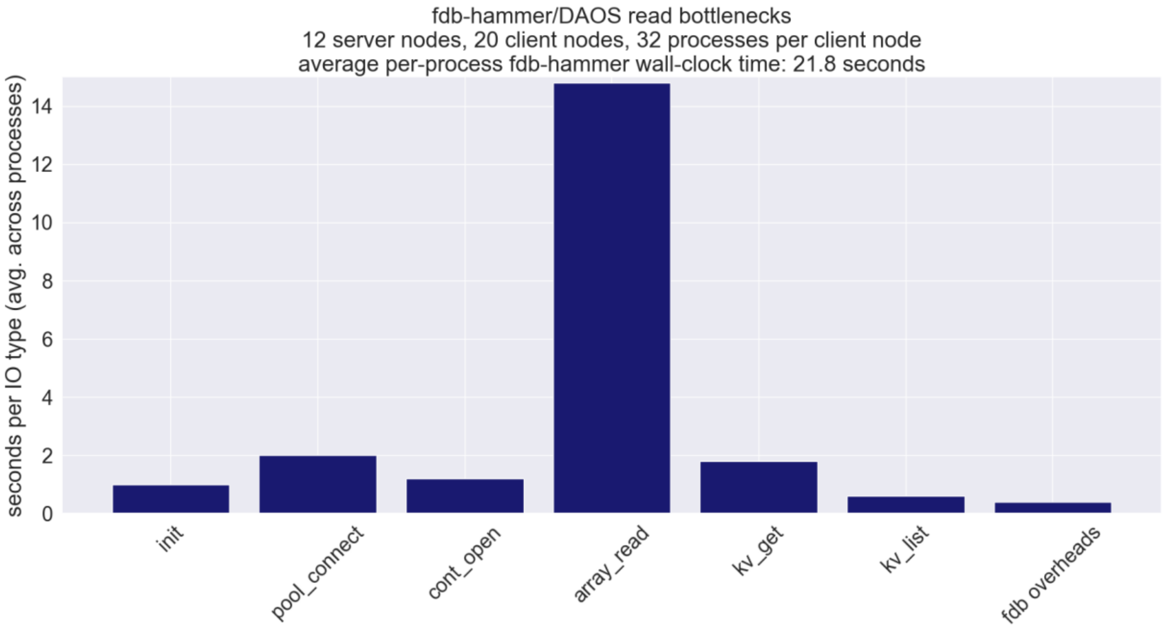


intel ECMWF Google Cloud epcc

Profiling

- Example breakdown of where time is being spent
  - Manual profiling





Approach/recommendations

- Key-Value contention
- For a specific benchmark run configured with contention across processes on indexing Key-Values:
    - 20 GiB/s write
    - 13 GiB/s read
- Tweaking the benchmark configuration to have all processes operate on a separate Key-Values:
    - 35 GiB/s write
    - 68 GiB/s read
- This may not be trivial or possible for all applications, but if design can achieve it then this improves performance

Approach/recommendations

- Avoid communications on/with the server where possible
- Cache objects locally in DRAM if possible
- Use `daos_array_open_with_attr` to avoid `daos_array_create` calls
  - Only supported for `DAOS_OT_ARRAY_BYTE`, not for `DAOS_OT_ARRAY`
  - Warning: the cell size and chunk size attributes need to be provided consistently on any future `daos_array_open_with_attr` to avoid data corruption
- `daos_array_get_size` calls can be expensive
  - Can store array size in our indexing Key-Values
  - Can manually calculate
  - Also possible to infer the size by reading with overallocation:
    - use `DAOS_OT_ARRAY_BYTE`, over-allocate the read buffer, and read without querying the size. The actual read size (`short_read`) will be returned
- `daos_cont_alloc_oids` is expensive, call it just once per writer process
  - Required to generate object ideas to use in calls but can generate many at one

Approach/recommendation

- Creating several containers (starting at ~300) in a DAOS pool reduces performance
- Opening the same container from all processes is expensive
  - this happens even if only a few containers exist in the DAOS pool
  - e.g. out of 20 seconds taken by a process to write 2000 fields, 1.5 seconds were spent just to open one container
  - we observed this starting at ~200 parallel processes
  - Sharing handles using MPI is the way to fix this
- Opening more than one container per process is very expensive
  - e.g. out of 30 seconds taken by a process to read 2000 fields, 6 seconds were spent just to open two containers

Approach/recommendations

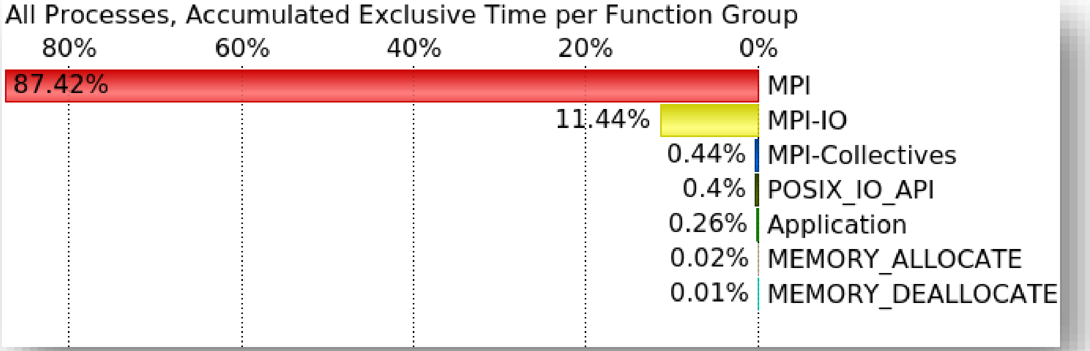
- `daos_key_value_list` is expensive
- `daos_array_open_with_attrs`, `daos_kv_open` and `daos_array_generate_oid` are very cheap (no RPC)
- Normal `daos_array_open` is expensive
- `daos_cont_alloc_oids` is expensive
- `daos_kv_put` and `_get` are generally cheap
  - Value size impacts this
- `daos_obj_close`, `daos_cont_close` and `daos_pool_disconnect` are cheap
- Server configuration to use available networks/sockets/etc… important for performance
  - Just like any storage system or application

intel ECMWF Google Cloud epcc

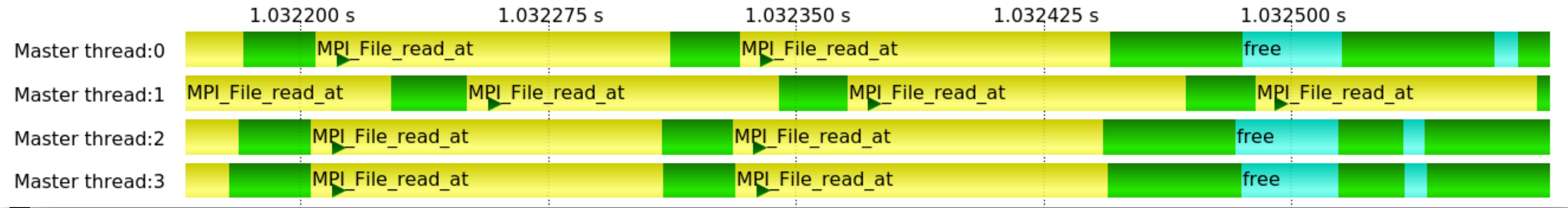
Object store usage design

- Mapping data structures to KV and Array objects is key to getting good performance functionality
- We suggest mapping contiguous chunks of arrays to be stored to single DAOS array object
  - Collect multiple arrays with associated KV to make the whole array
- Can be as extreme as having a single value per KV
  - Significant overheads in this
- Depends on your application data structures you may want to aggregate less data for I/O
  - Group based on meaningful/scientific dimensions
- HDF5 or similar hierarchies could map well to Keys with Arrays
- Adding keys to the array data/values can let data set structure to be created, enumerated, and extended
- See the Exercises/FullApplication in the GitHub repository for the tutorial

Summary



- Object storage can provide high performance
  - DAOS: 90+ GB/s per server is possible
  - Hardware and configuration dependent, just like all I/O
- Built in replication and redundancy under your/user control
- Different interfaces available
  - Filesystem for zero cost porting
  - Simple file like access for slightly improved performance at little effort
  - Programming APIs for full functionality
- Object store interface enables changing I/O granularity/patterns for bigger benefits



Final Summary

- Thanks for attending!
- We're keen for feedback
  - Can provide through the digital experience
    - "**give feedback**" button under Event Type in the Digital Experience
  - Can also provide directly (a.jackson@epcc.ed.ac.uk)
- Happy to take further questions when/if they occur to you
  - Email or come and talk to us
- Tutorial system will stay active for the week
  - Time to complete the exercises/experiment with the technology
  - Any problems email me as well
- Want more help
  - Come and speak to us
  - Happy to collaborate/help with object store usage/porting/etc...