Jakob Dieterle, Julian Kunkel, Hendrik Nolte

Scalable Software Distribution for HPC-Systems Using MPI-Based File Systems in User Space

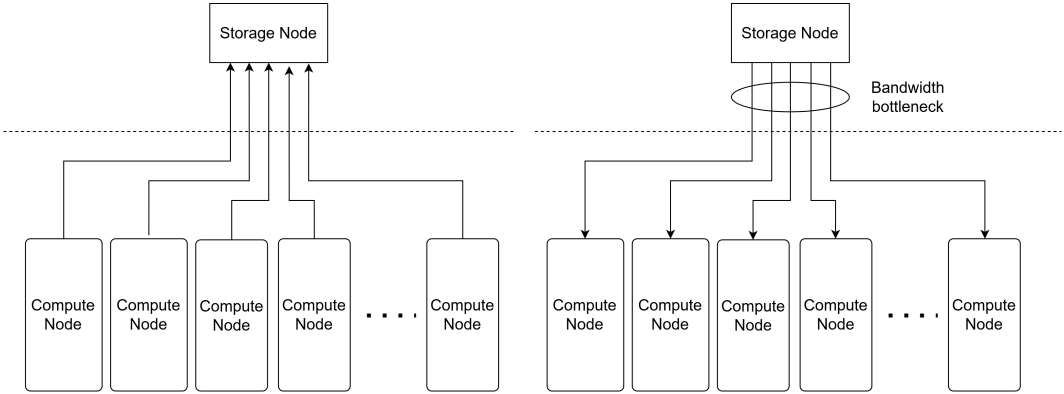
Presenter: Jakob Dieterle, GWDG, jakob.dieterle@gwdg.de

Presenter - Jakob Dieterle

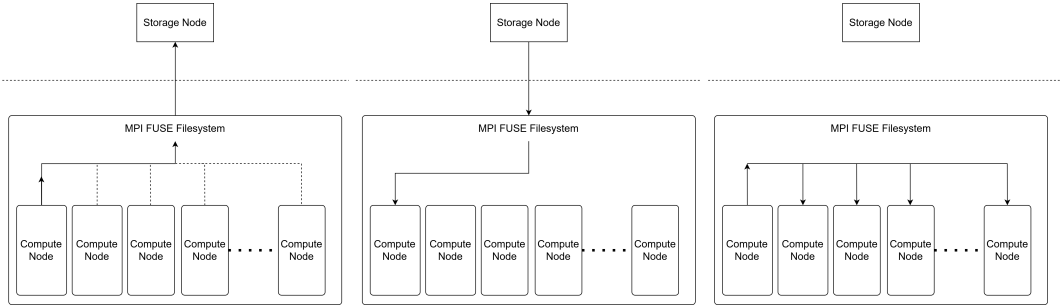
- 2018-2024: BSc applied computer science, Georg-August-Universität, Göttingen
- Publications:
  - Kqiku, Lindrit; Dieterle, Jakob; Reinhardt, Delphine, Exploration of a Mobile Design for a Privacy Assistant to Help Users in Sharing Content in Online Social Networks In *Mensch und Computer 2022*, Darmstadt, 2022.
- Work experience (student assistant):
  - 2024-now: GWDG
  - 2022-2024: The German Aerospace Center (DLR)
  - 2020-2022: Georg-August-Universität Göttingen

Table of contents

Problem Outline



Solution Outline



Tools: MPI

MPI communication concepts:

- Point-to-Point communication
  - Send / Recv

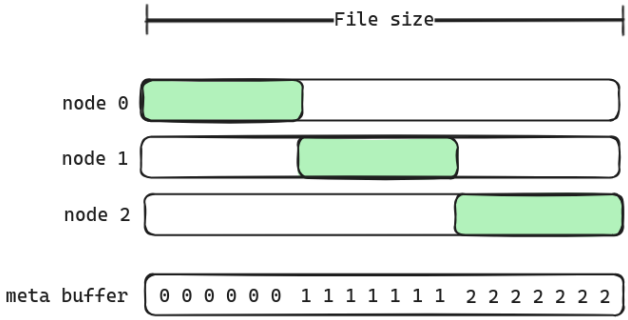
  → A lot of orchestration needed (overhead), blocking
- Collective communication
  - Broadcast / Gather / Allgather

  → Problem: nodes want to access different parts of the file, blocking
- One-Sided communication
  - Put / Get on MPI_Windows

  → Allows asynchronous and non-blocking communication

Design: One-Sided Reading

Idea: distribute file during open, read from other node's buffers



Design: One-Sided Reading

Open method

```
def my_open(path):
  file = open(path)

  file_buf = MPI_Win_allocate(..)
  meta_buf = int[file.size]

  meta_buf = get_distribution(..)

  offset, size = get_my_range(..)
  data = read(offset, size)
  file_buf[offset:offet+size] = data

  return file_handler
```

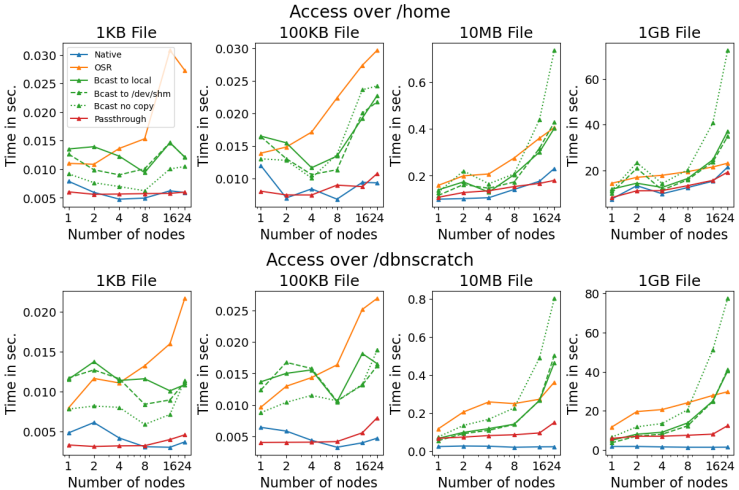
Open method

```
def my_read(file, range):
  if (in_buffer(range)):
    return file_buf[range]

  targets = get_targets(range)
  for target in targets:
    t_range = get_target_range()
    data = MPI_Get(t_range ,target)
    file_buf[t_range] = data

  meta_buf[rage] = my_rank

  return file_buffer[range]
```

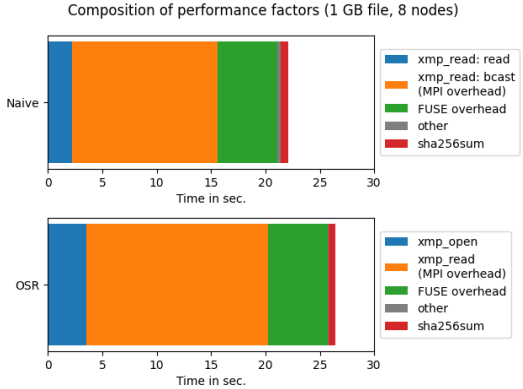
Methodology

- Metric: timings of sha256sum on test file
- Tests run on Sofja
  - home file system
  - scratch file system
- 1 to 24 nodes
- 1KB - 1GB randomized test data
- 10 runs per configuration

Results: Time to compute hashsum of random test data



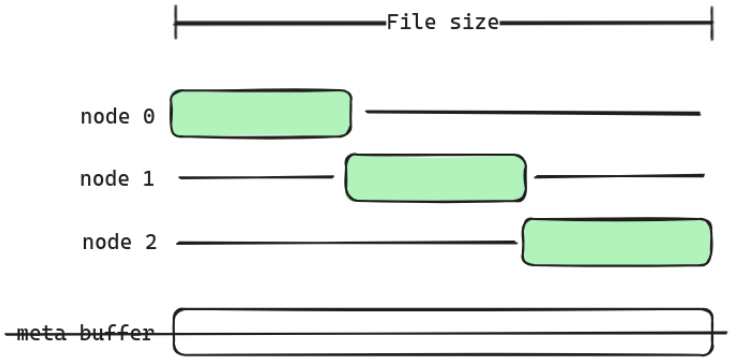
Results: Factor Analysis



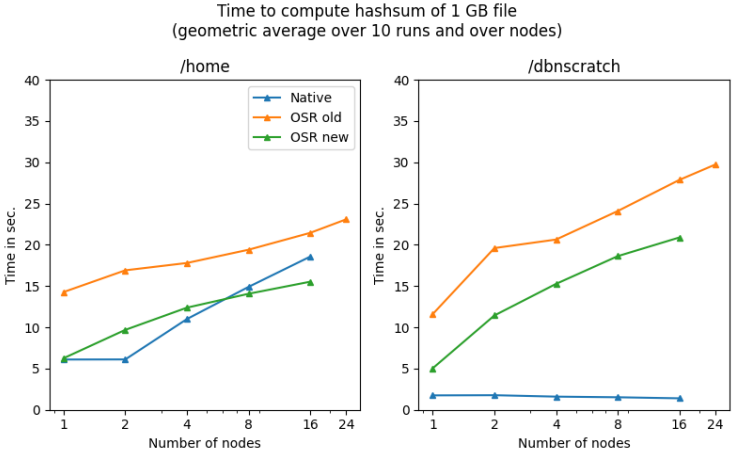
- Operations on meta buffer alone take a lot of time
- Buffer has no positive effect in certain use cases

  $\rightarrow$ test updated version with without local buffer

Updated OSR



Results: Updated OSR



Main Takeaways

- FUSE adds significant overhead
  - grows with number of read requests (file size)
  - unchanged with varying number of nodes
- MPI also adds significant overhead
  - grows with number of communication calls (file size and number of nodes)
- OSR can have advantage over slower file systems
- Optimized file systems are much faster on the scale of the tests
- Bandwidth usage reduced to a minimum

Future Work

- Run tests on larger scale (Emmy)
- Test different metrics / use cases
- Improve OSR design:
  - Use `MPI_Win_lock_all` to reduce MPI overhead
  - Develop more advanced caching mechanism
  - Allowing FUSE to run multi-threaded
  - Handling multiple files at once