Log-structured files for fast checkpointing

Milo Polte

Jiri Simsa, Wittawat Tantisiriroj,Shobhit Dayal,

Mikhail Chainani, Dilip Kumar Uppugandla, Garth Gibson

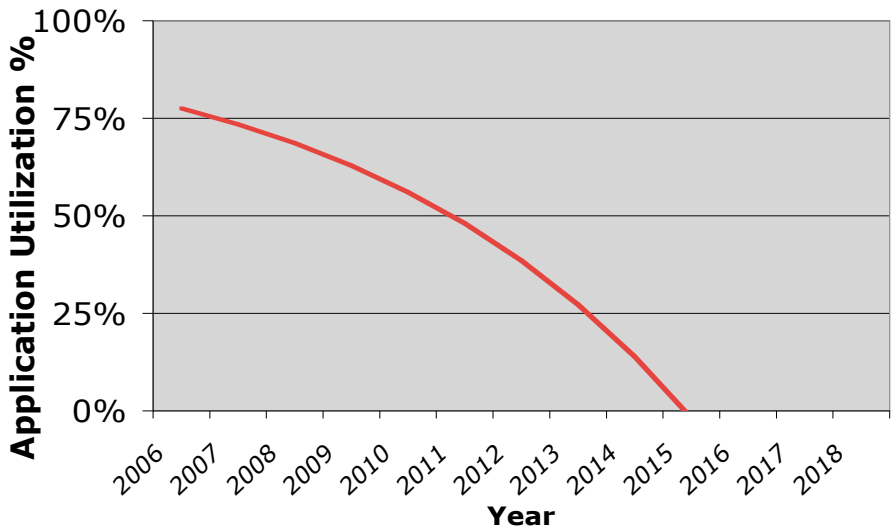
PARALLEL DATA LABORATORY

Carnegie Mellon University

Motivation

- HPC systems growing larger
  - More nodes
  - Larger memory to checkpoint
  - More frequent failures
- Application effectiveness means more frequent, larger checkpoints in same total writing time
- Solutions?
  - Fewer failures
  - Faster checkpoints

For this talk:
Want an application generic solution without special hardware



Checkpointing

- Clients stop work, barrier sync, write state to shared storage
- Large scale
  - Thousands of nodes
  - Terabytes of data
- No app work until checkpoint complete
  - Need to be done as quickly as possible
- Two possible strategies:
  - N-to-N Checkpointing
  - N-to-1 Checkpointing

N-to-N Checkpointing

- In N-to-N checkpointing, one file per client
  - No per-file writing lock contention
- Disk still sees seeks from multiple writers
  - Bad disk spindle utilization
- Lots of metadata creates
- Potentially difficult to manage

N-to-1 Checkpointing

- All clients write their state to a single file in shared storage
- Little metadata traffic
- Lock conflicts and many seeks
  - Bad disk spindle utilization
- Some apps do strided small writes from all nodes
  - More lock contention
  - Seeks?

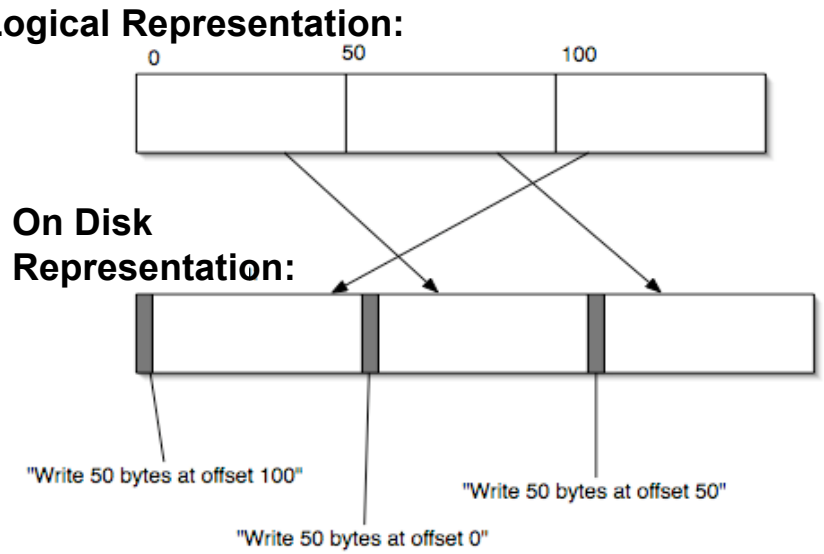
- What this talk is about

Log-Structured Writing

- A log can allocate random writes sequentially
  - Reduces number of seeks
- Proposed in LFS (Rosenblum, Ousterhout, 1992)
  - Implemented in WAFL and PanFS
- Our use for checkpoints inspired by Zest

Series of writes:

```
pwrite(file, buffer[100], 50, 100);
pwrite(file, buffer, 50, 0);
pwrite(file, buffer[50], 50, 50);
```

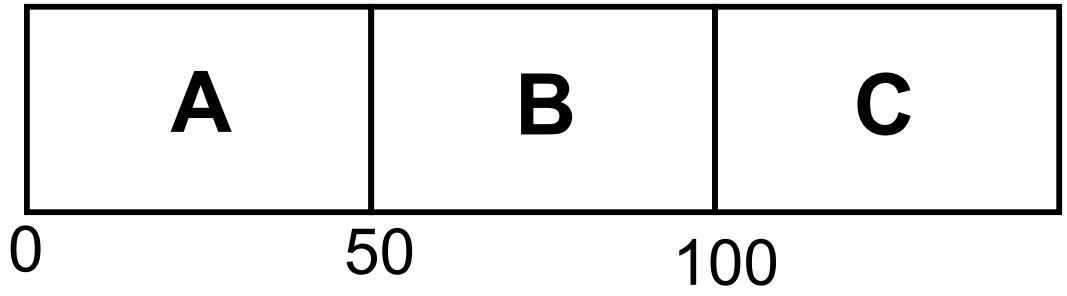


Log-Structured Writing

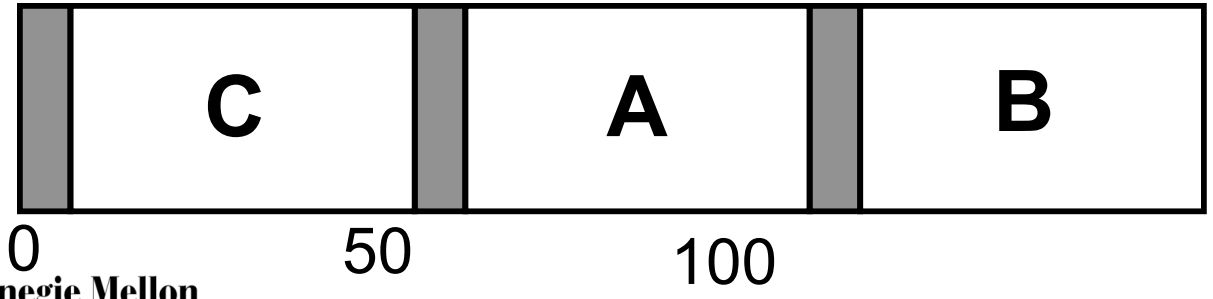
Series of writes

```
pwrite(file, buffer[100], 50, 100);
pwrite(file, buffer, 50, 0);
pwrite(file, buffer[50], 50, 50);
```

In memory buffer



On disk representation



Log structured problems

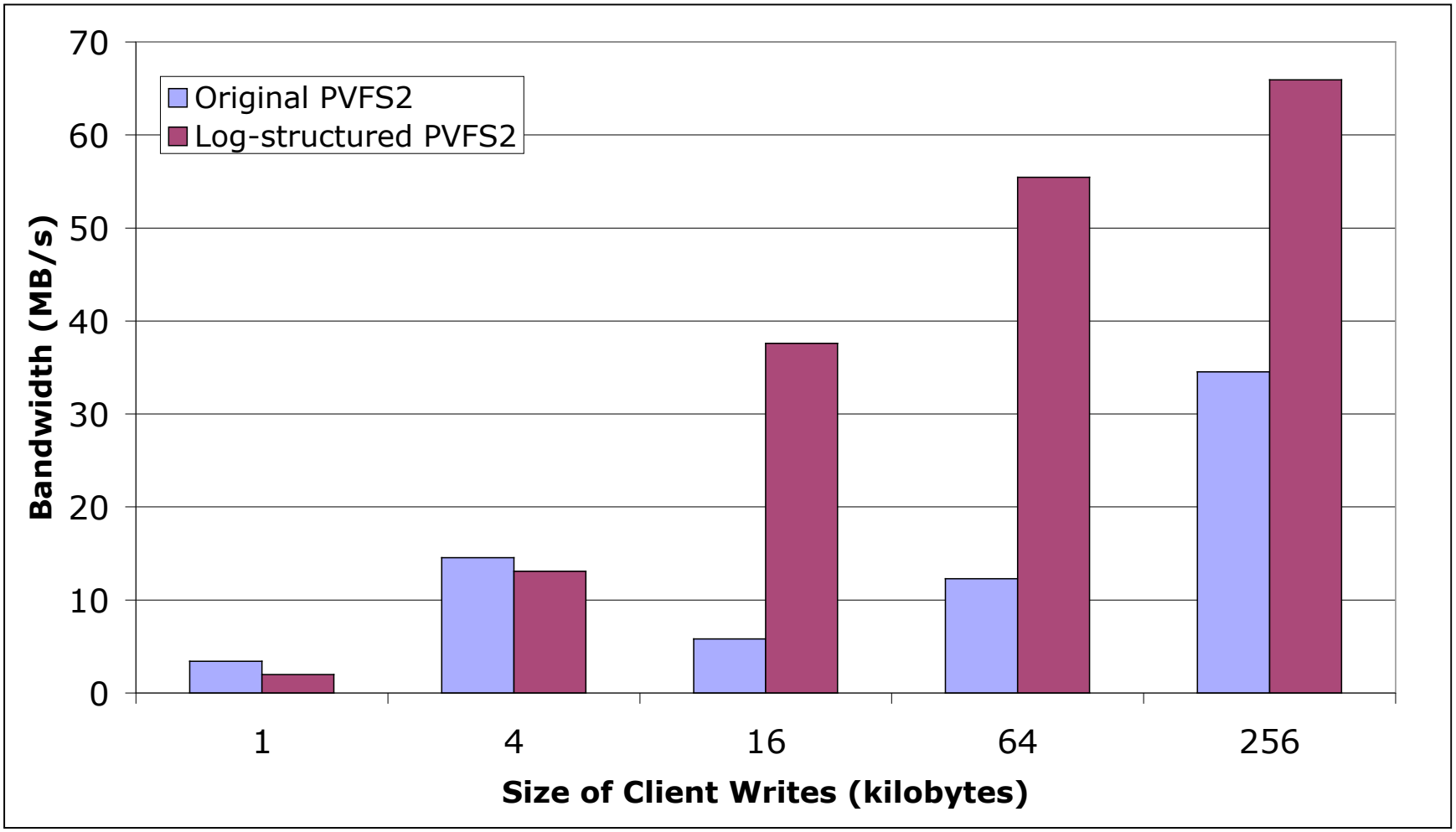
- Can struggle with write random, read sequential
- But checkpoints are “write-once read-maybe”
  - Hopefully….
- Log writing may be inappropriate for entire filesystem
- Want option for per-file customized representation

- Our idea: Log writing on a per-file representation in a distributed filesystem
  - Assigned as class project in Advanced Storage Systems

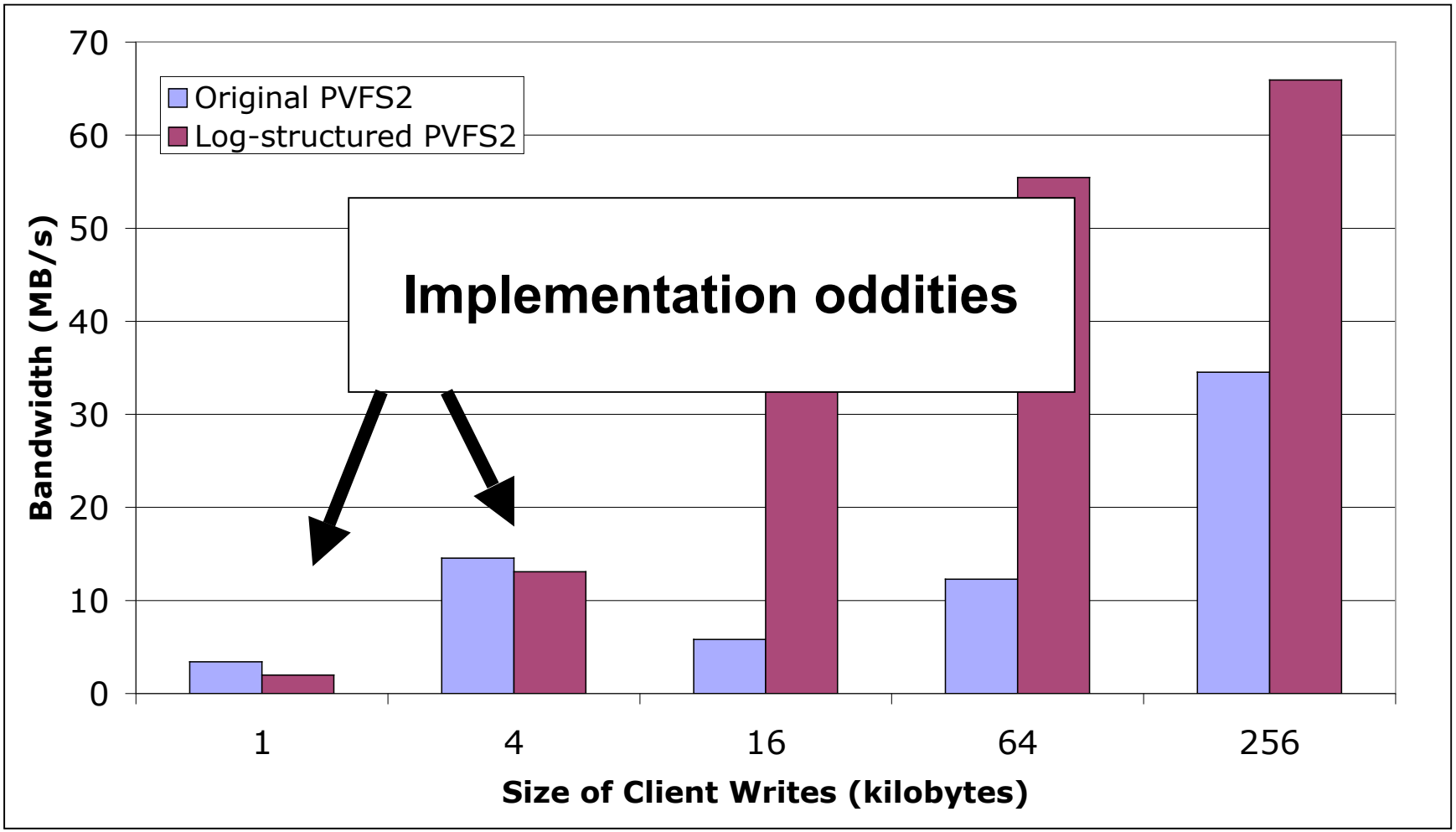
Experimental Design

- Implemented log writing in PVFS2
  - A distributed parallel filesystem running on Linux
  - Storage server modified to write files in log fashion
  - Reads scan file for all headers to find latest copy of data
  - Could have been implemented at different level
    - Library
    - App
    - But filesystem considered most universal
- Application: `mpi_io_test`
  - Benchmark from LANL for an N-to-1 checkpoint
- Preliminary evaluation of 10 clients, single server

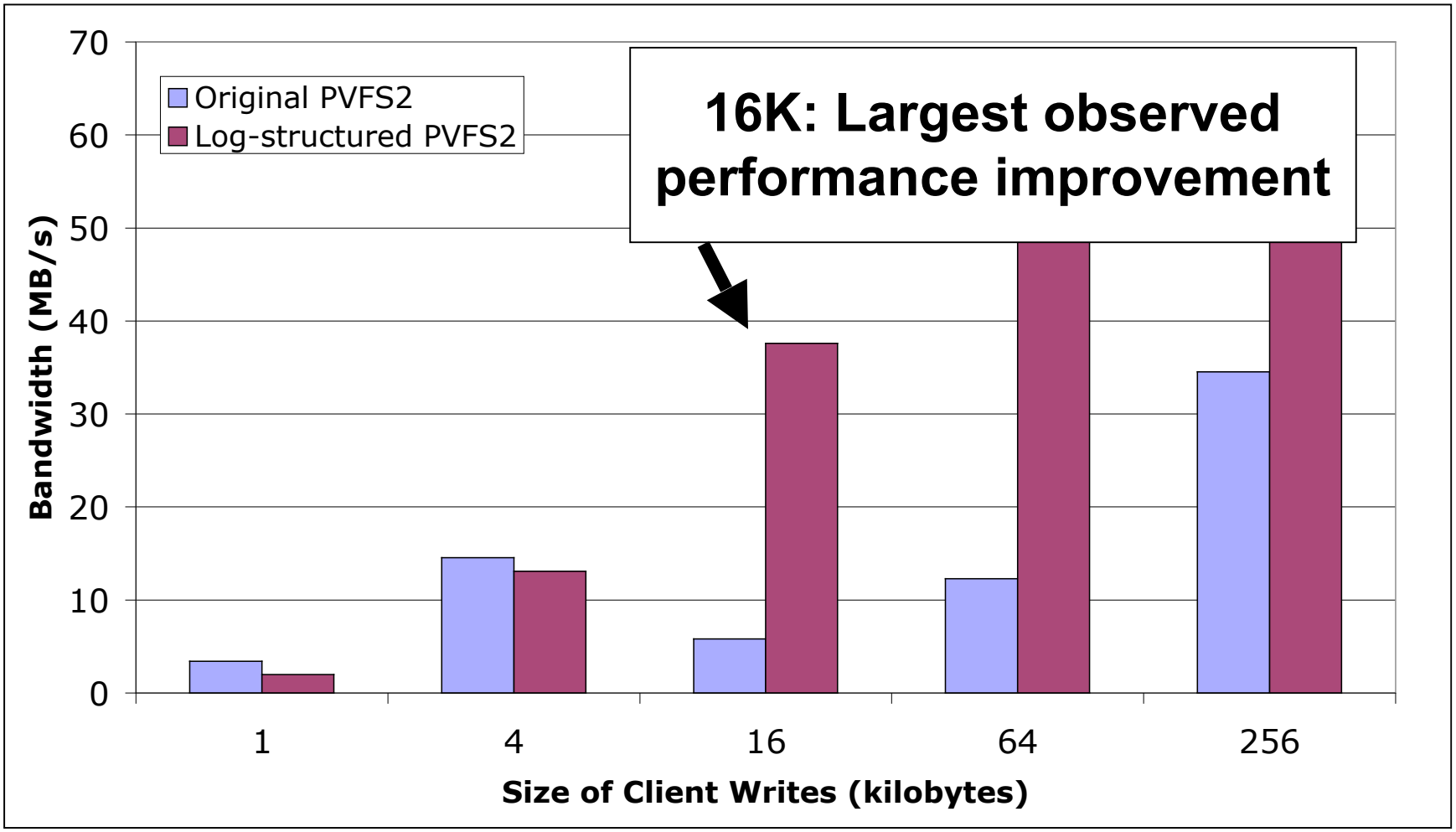
Results - Writes



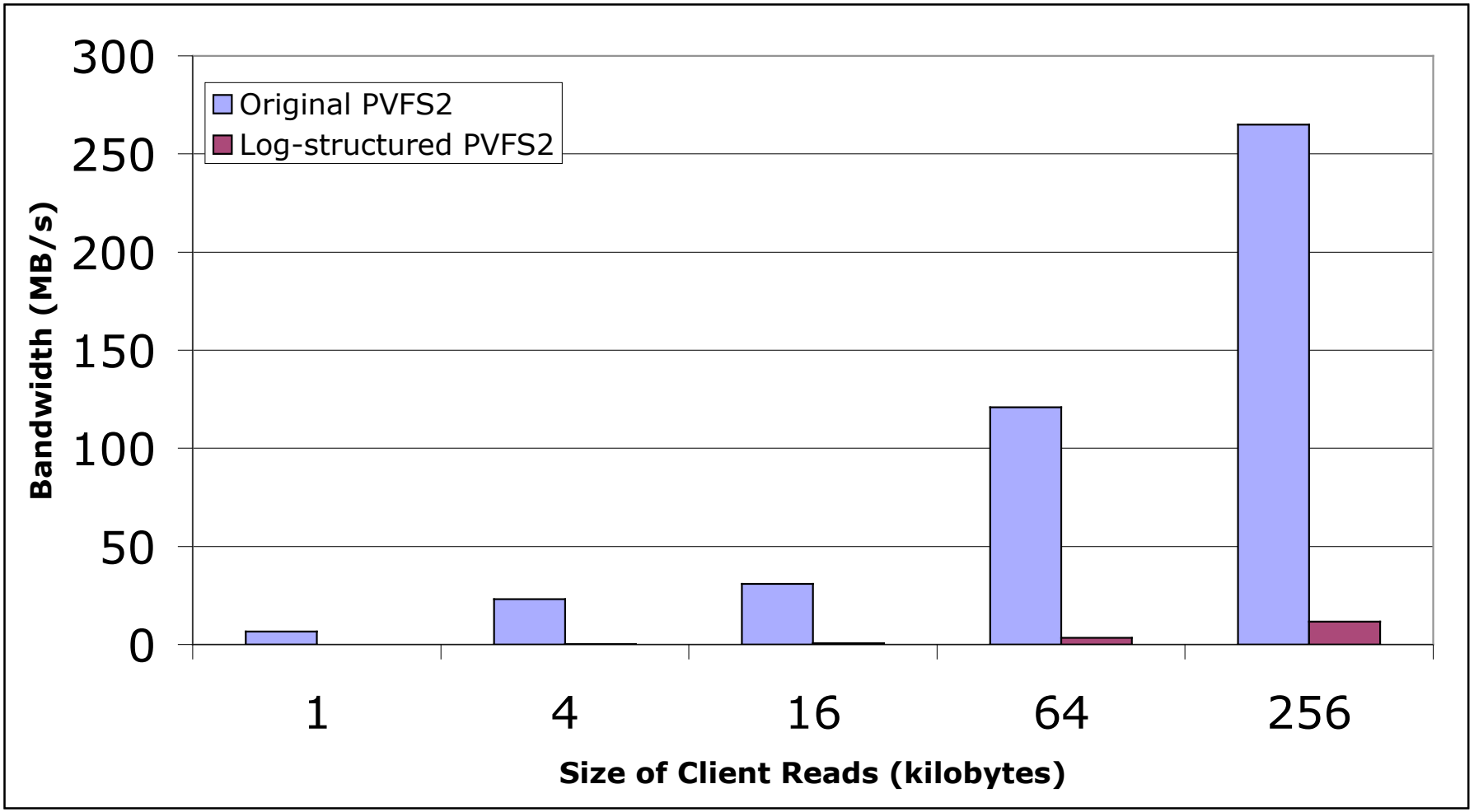
Results - Writes



Results - Writes

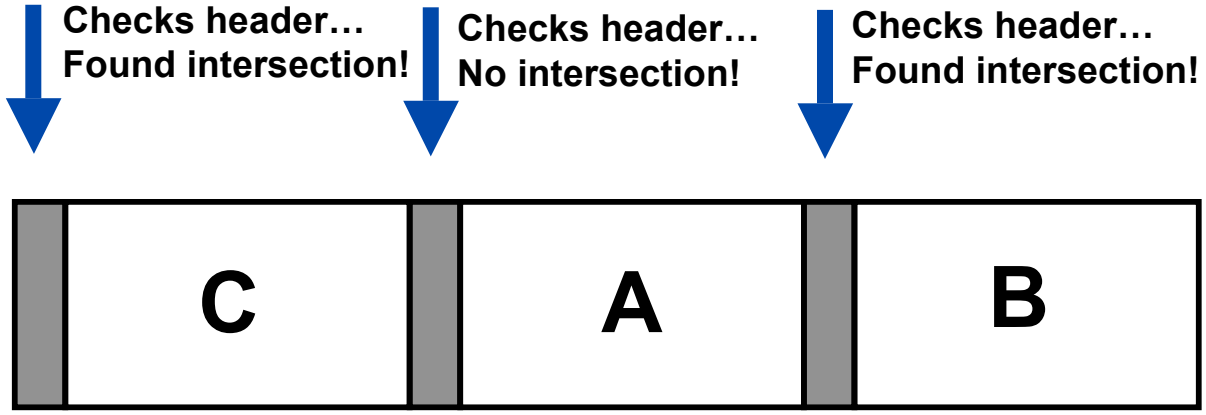


Read Performance (oops!)



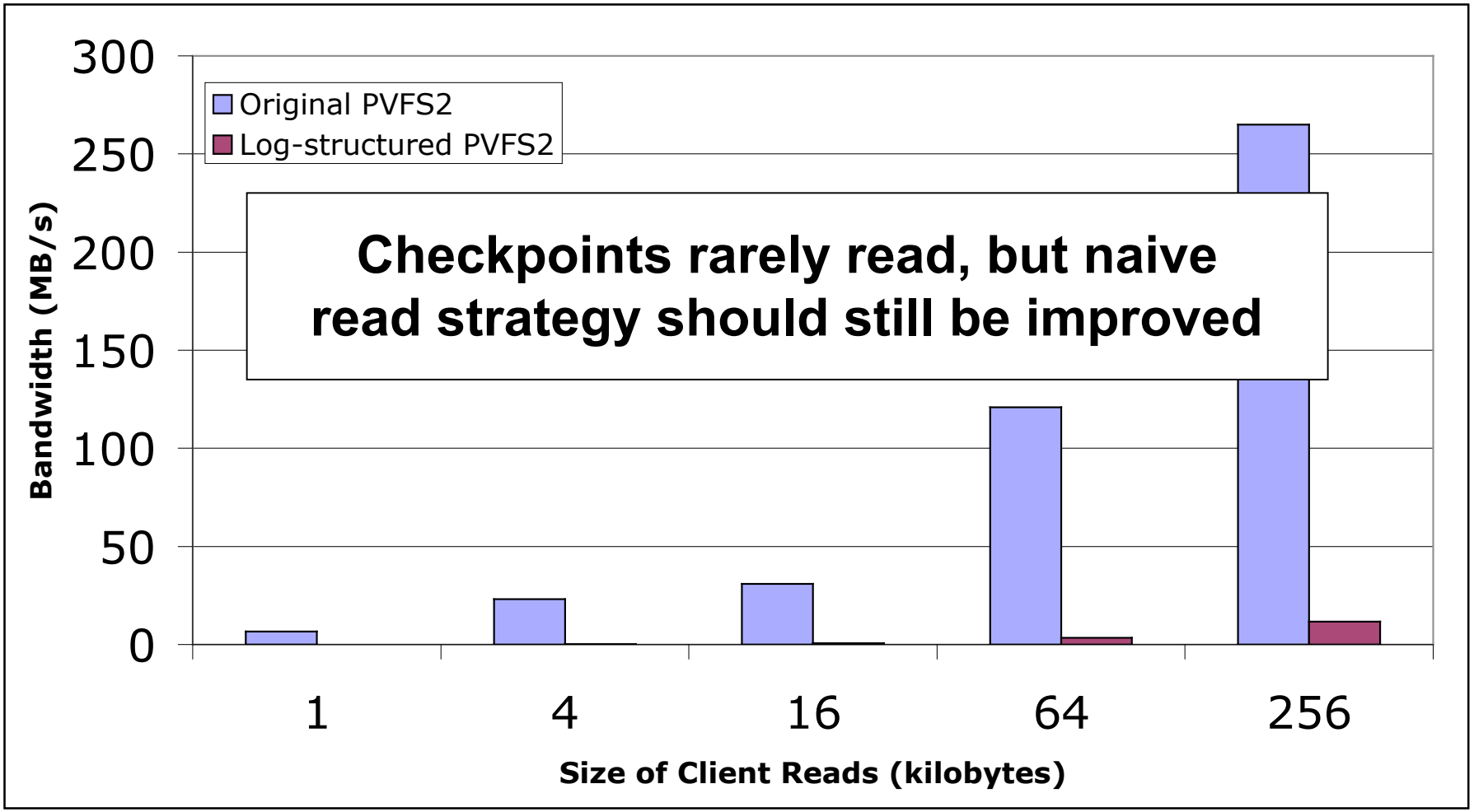
Read Performance (oops!)

- Why is read performance so bad?
- EACH read scans ALL headers



- Student project naïve implementation
- Better ideas:
  - Footers rather than headers for backwards file scan
  - Flatten the entire file on the first read
  - Keep the index in separate metadata for faster scans

Read Performance (oops!)



Conclusions and Future Work

- Class project results promising potential benefits for per-file logging on a distributed filesystem
- Read path could still use improvement
- What about other per-file representations?
  - Per-file RAID (like PanFS)
  - Optimized format for scientific data files (e.g. NetCDF)?
  - Other indices
- Per-client logging?
  - N-to-N checkpointing behind N-to-1 interface
  - No lock contention, false sharing