Self-Adaptive Hints for Collective I/O

Joachim Worringen

Work Done at C&C Research Labs, NEC Europe Ltd.

Now with 

Interconnect Solutions

<joachim@dolphinics.no>

MPI-IO File Hints

- File hints are used to tell the MPI-IO library
  - which kind of access pattern to expect (*high-level*)
  - which feature to use with which parameter (*low-level*)
- File hints are are passed to the library as MPI_Info objects: <key, value> tuple
- Some file hints are defined in the standard:
  - High level (*only one!*):
    - **`access_style` can be set to** `read_once`, `write_once`, `read_mostly`, `write_mostly`, `sequential`, `reverse_sequential`, `random`
  - Low level (*many!*):
    - `cb_block_size`, `io_node_list`, `cb_buffer_size`, `chunked_size`, ...

MPI-IO library is free to ignore any file hint, or use its own!

Motivation

- **MPI-IO file hints are a useful technique!**
  ... but nobody uses them in the „real world“.

  Why?
  - „Never heard of this – file hints?“
  - „Does this really work?“
  - „I tried and it did not change anything.“
  - „I tried and performance degraded on another machine.“
  - „I don't know which one to use, and why.“
  - „MPI-IO?“

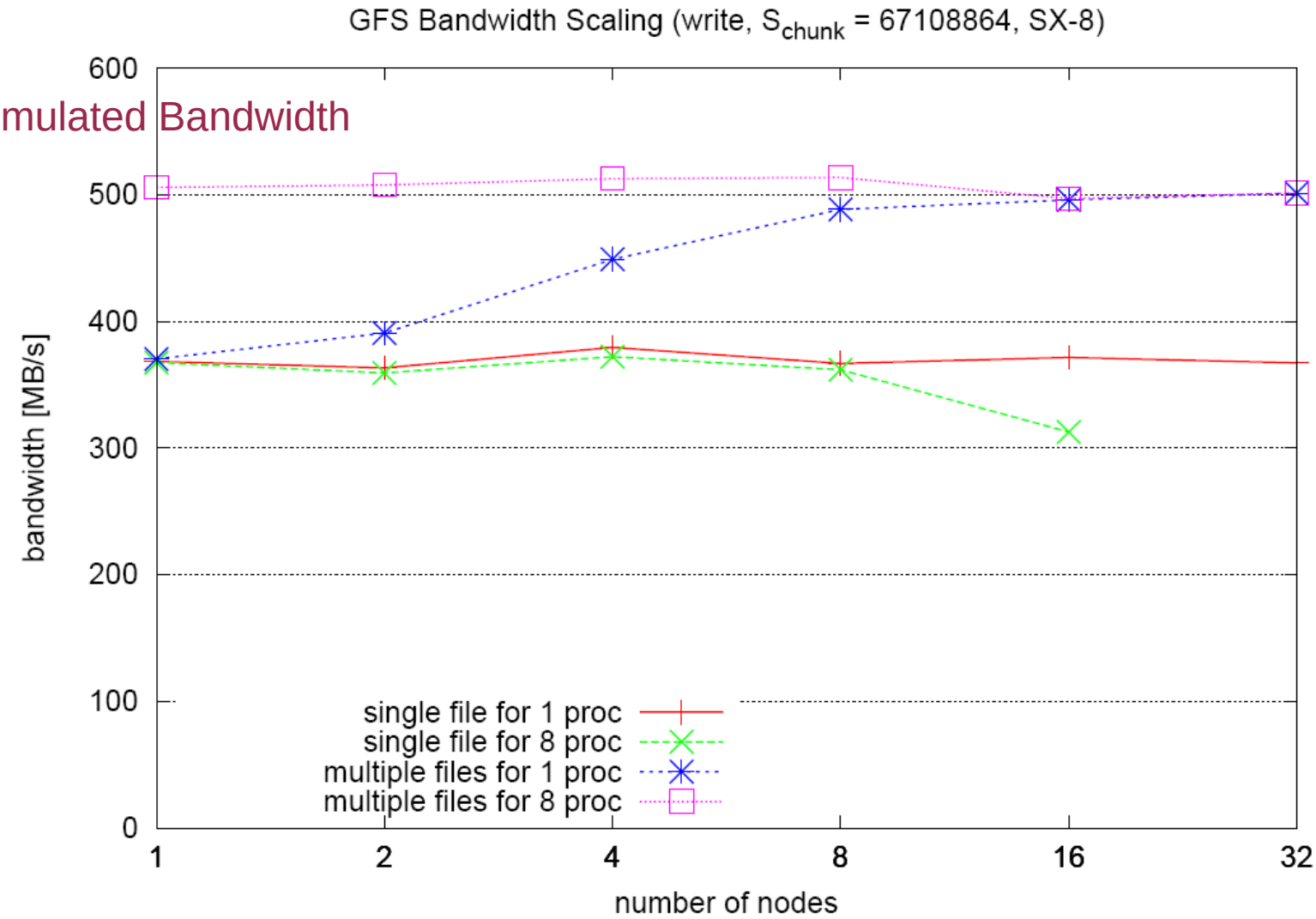
I/O on NEC SX Systems

- NEC uses GFS as cross-platform file system
  - Uses high-performance, highly reliable FibreChannel storage
  - Can be accessed with every NFSv3 client
  - GFS clients use NFS below 64kB, and *third-party-transfer beyond*
    - *Avoid file-server bottleneck for streaming data access*

- *NFS „semantics“ sub-optimal for parallel I/O:*
  - *Explicit locking & flushing required for multi-node access to ensure consistency*
    - *Effectively, no caching takes place*
  - *Other bottlenecks in storage path do occur if many processes access a single file*

GFS Scaling Characteristics

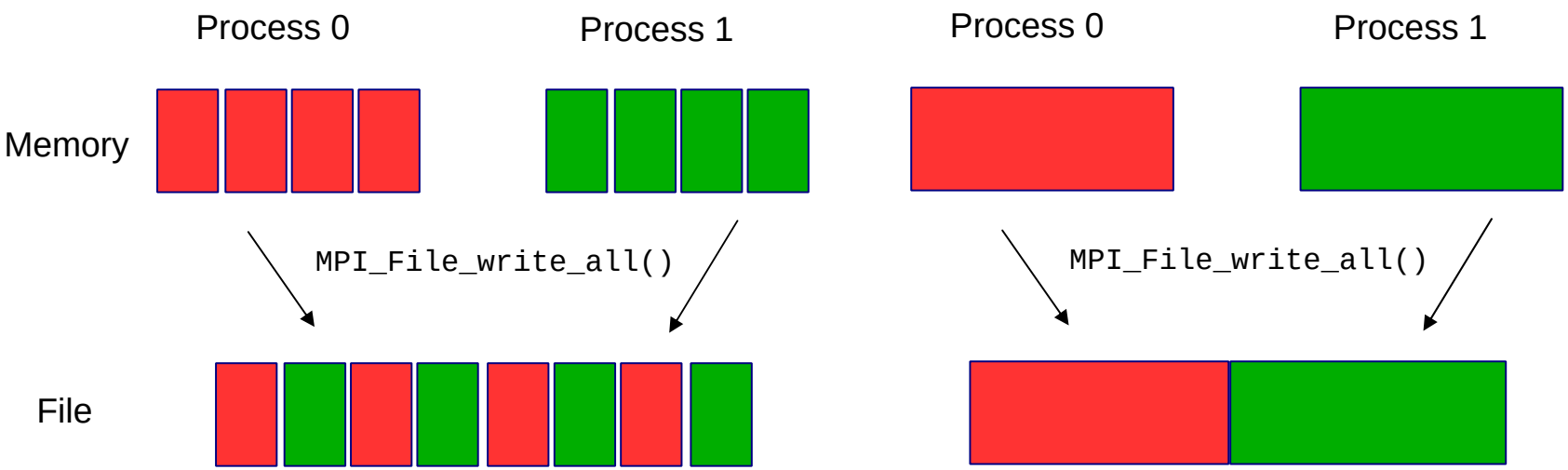
Accumulated Bandwidth



Data shown not representative for any NEC GFS performance – highly dependant on chosen configuration of storage system!

The Trigger Case

- b_eff_io benchmark on NEC SX-8 showed performance drop for one test of *scattered* pattern
  - *scattered* performs **non-contiguous** file access
    (which is highly optimized in MPI/SX – see EuroPVM/MPI 2003)
  - Problematic test actually performs **contiguous file access!**
    - **Most tests: memory buffer size != file block size**
    - **This test: memory buffer size == file block size**



Where's the Problem?

- MPI-IO detects that accesses are not overlapping
  - ➔ Individual I/O is considered „good enough“:
  - ➔ Each process writes its 1MB itself

  Reduced I/O size + increased locking overhead

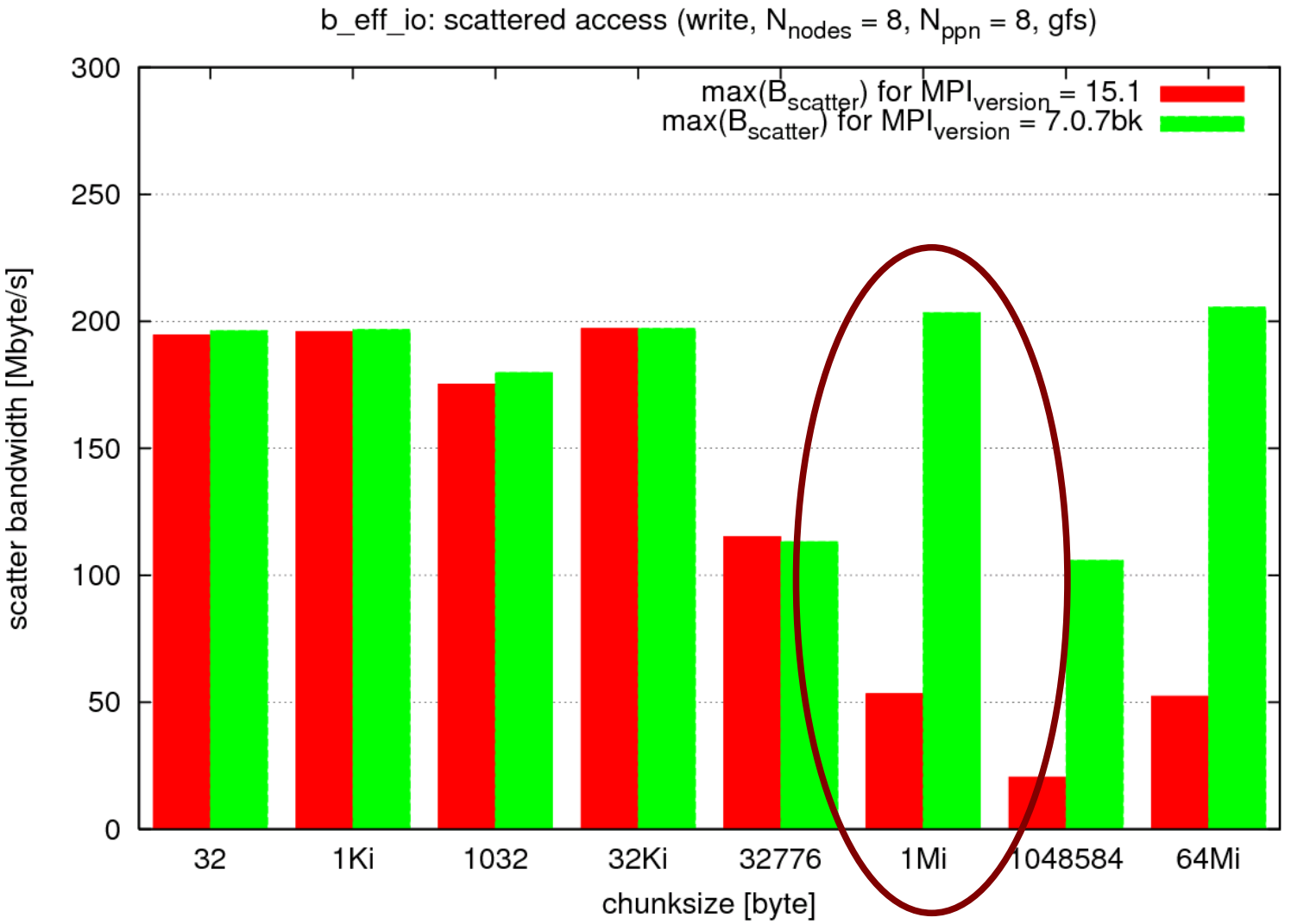
  = performance degradation
- What we want instead:
  optimized collective two-phase I/O with collective buffering
  - Can be enforced by providing file hints that control *collective-buffering*:

    `cb_read` and `cb_write` both set to `enable`

- This solves *this problem (see next slide)...*

  ...but creates others in the same benchmark run.

Performance Improvement



A Real Solution?

- We want to achieve the same performance improvement

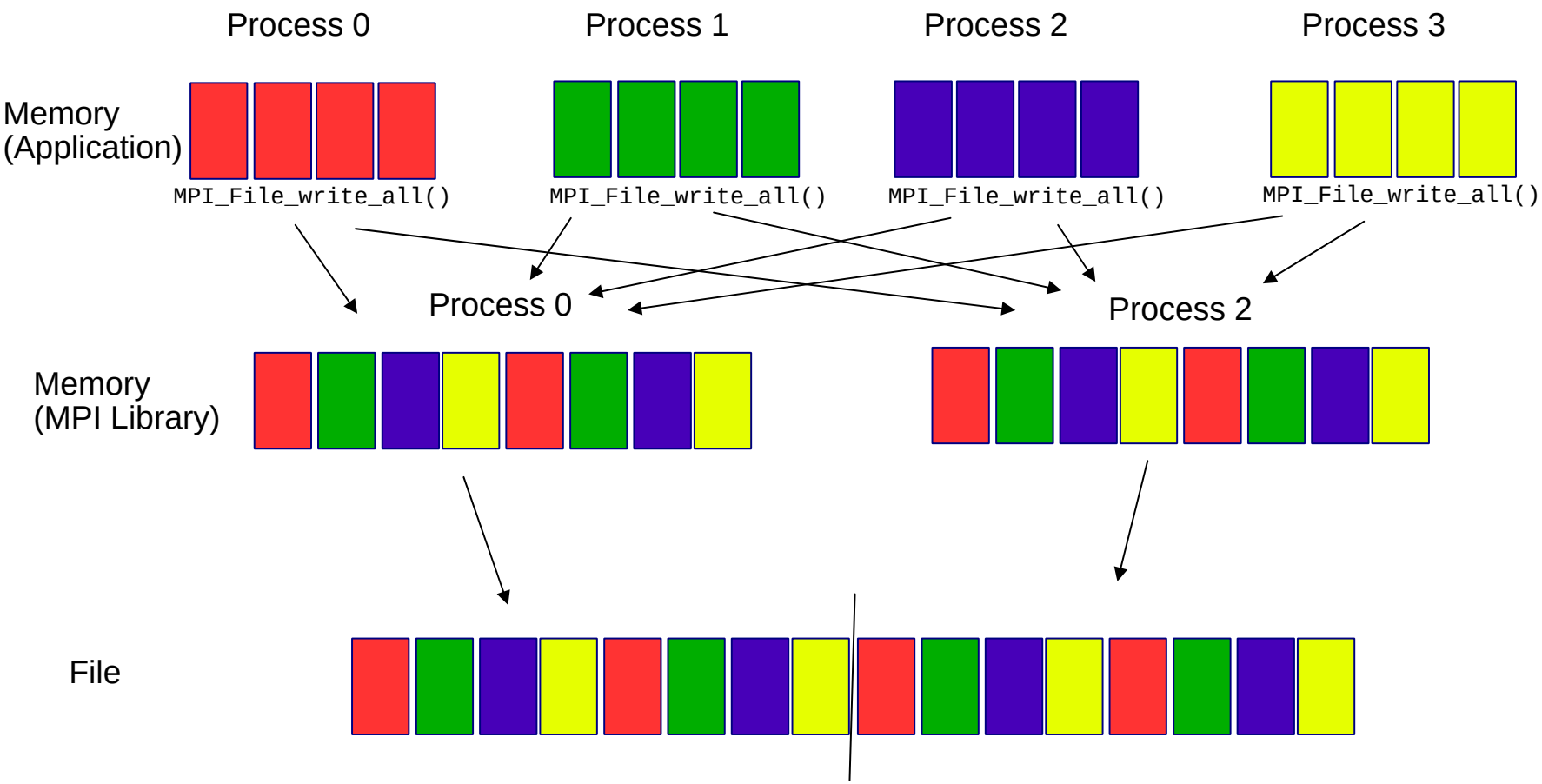
  without providing file hints

  ... and without the drawbacks!

  ➔ **A static solution is not possible. We need adaptivity!**

cb_procs

- `cb_procs` (aka `cb_nodes`) specifies the number of processes used for collective buffering:

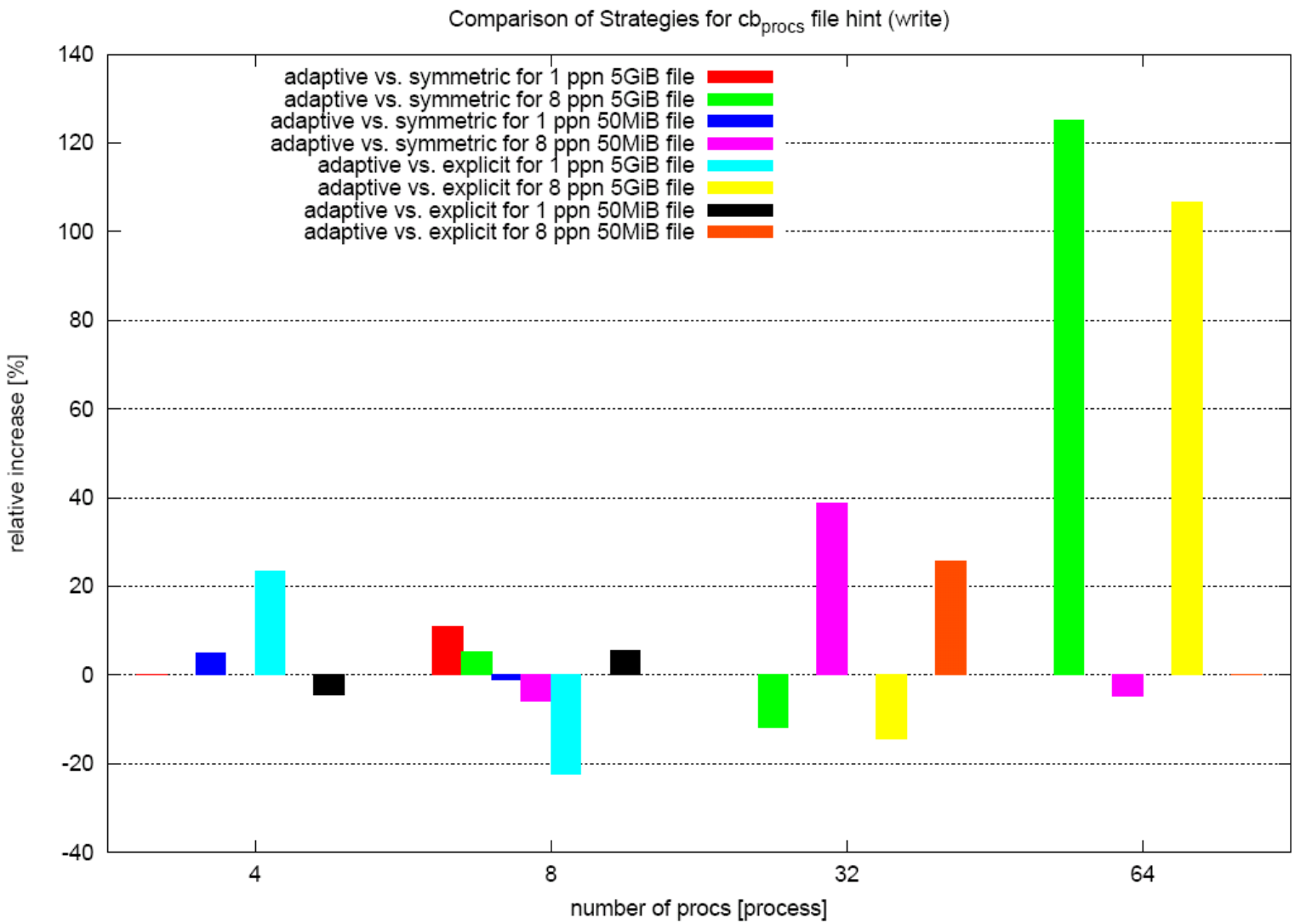


Adaptive cb_procs

- Use **minimal file domain size**
  - Reduces number of APs for small access sizes
- Use **a lower bound for number of APs** relative to the total number of processes
  - Do not limit concurrency too much
- Use **at most 4 APs** per node (for SX with 8 CPUs/node)
  - Do not „oversubscribe“ the I/O interface

- Align file domain bounds to NFS block size to avoid lock conflicts

Improvement



Adaptive cb_config_list

- `cb_config_list` allows to specify an explicit list of nodenames to operate the APs on.
  - Does not make sense on symetrically configured nodes
- **Strategy for MPI/SX:**
  - Determine number of APs
  - Place these APs round-robin on the available nodes

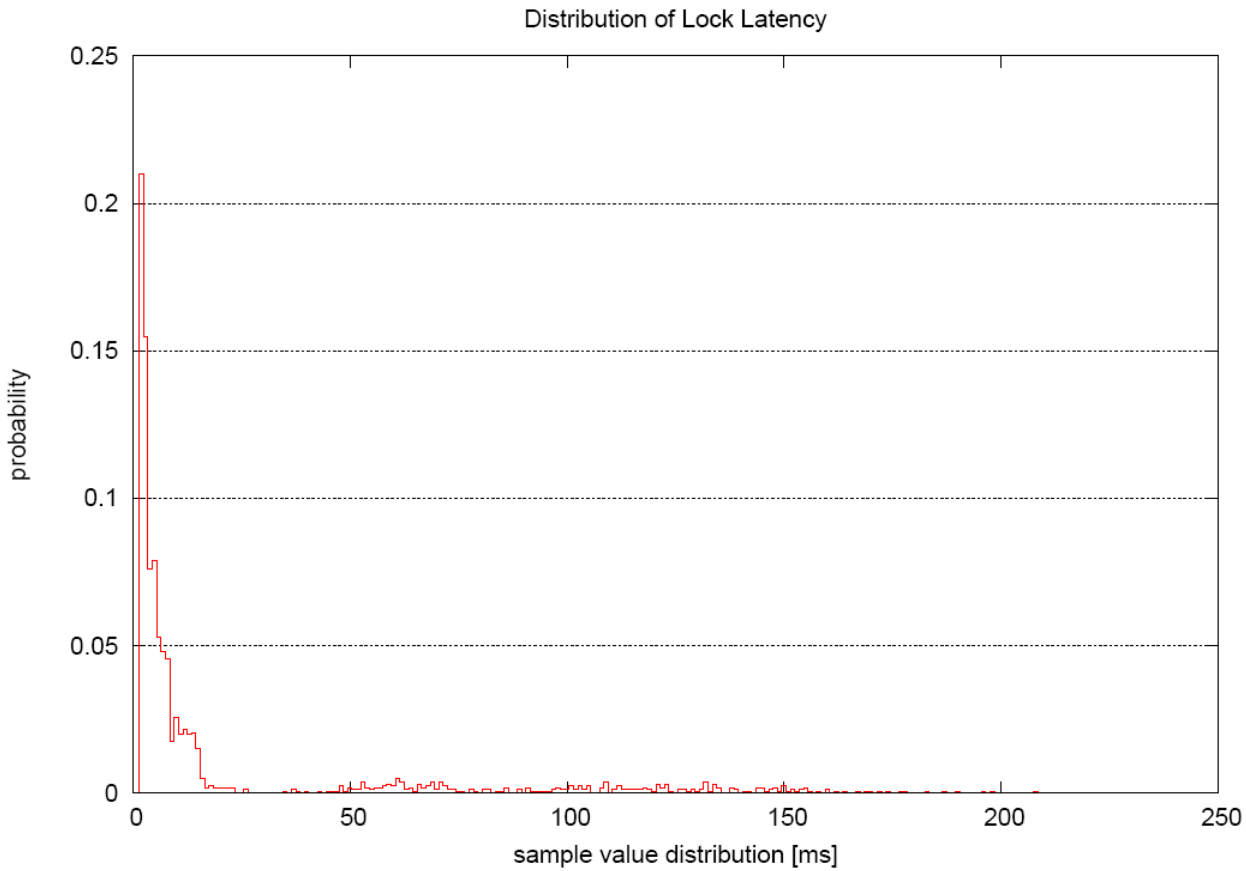
Adaptive cb_read / cb_write

Very delicate hint:

- Using it incorrectly has significant performance impact
- User needs to have detailed knowledge to set it correctly
- Automatic mode does not always give optimal results:
  - **Potential problems with individual file access:**
    - Many small accesses instead of few large ones
    - NFS locking conflicts
  - **Potential problems with (enforced) collective file access:**
    - Increasingly ineffcient for non-adjacent/non-overlapping access ranges
- **Strategy in MPI/SX:**
  - Do not decide on **„overlap or not“**, but determine a **„density factor“** (ratio of gaps to data within the access range)
    - Efficiently possibly with *list-less I/O*

Improvement

- Writing of adjacent blocks:
  - 16 processes call MPI_File_write_all() for 1500 bytes
- ROMIO default (individual I/O) vs. adaptive setting:
  - Lock latency 99th quantile: 160ms vs 5ms



Adaptive cb_buffer_size (read/write)

- Size of buffer for two-phase collective buffering:
  - Too small a buffer will limit I/O performance (GFS!)
  - Too large a buffer will affect application (memory) performance
- **Strategy for MPI/SX:**
  - Start buffer size at 1MB
  - Duplicate buffer size
    - Measure time for complete I/O operation
    - Bandwidth increased?
      - Yes: leave buffer size
      - No: half buffer size; stop increasing
  - Reset buffer size if
    - I/O size changes significantly
    - a new file view is established

Improvement

- Static Buffer of 8MB vs. adaptive buffer 2...128MB

	Single Process	64 processes on 8 nodes
Static	204 MB/s	205MB/s
Adaptive	298 MB/s	354MB/s

Adaptive buffer size typically ended at 32MB.

Summary

- High-level file-hints make sense

  ... but are often expressed via open-mode and UNIQUE_OPEN flag

- Low-level file hints are not used

  ... and can be set adaptively with some additional effort