FAST '08
6th USENIX Conference on File and Storage Technologies
USENIX
FEBRUARY 26–29, 2008 | SAN JOSE, CALIFORNIA
Sponsored by USENIX in cooperation with ACM SIGOPS, IEEE Mass Storage Systems Technical Committee (MSSTC), and IEEE TCOS

Computer Architecture and Systems Group
Department of Computer Science
University Carlos III of Madrid
Fco Javier García Blas, Florin Isaila & Jesús Carretero

View-based collective I/O for MPI-IO

View-based I/O

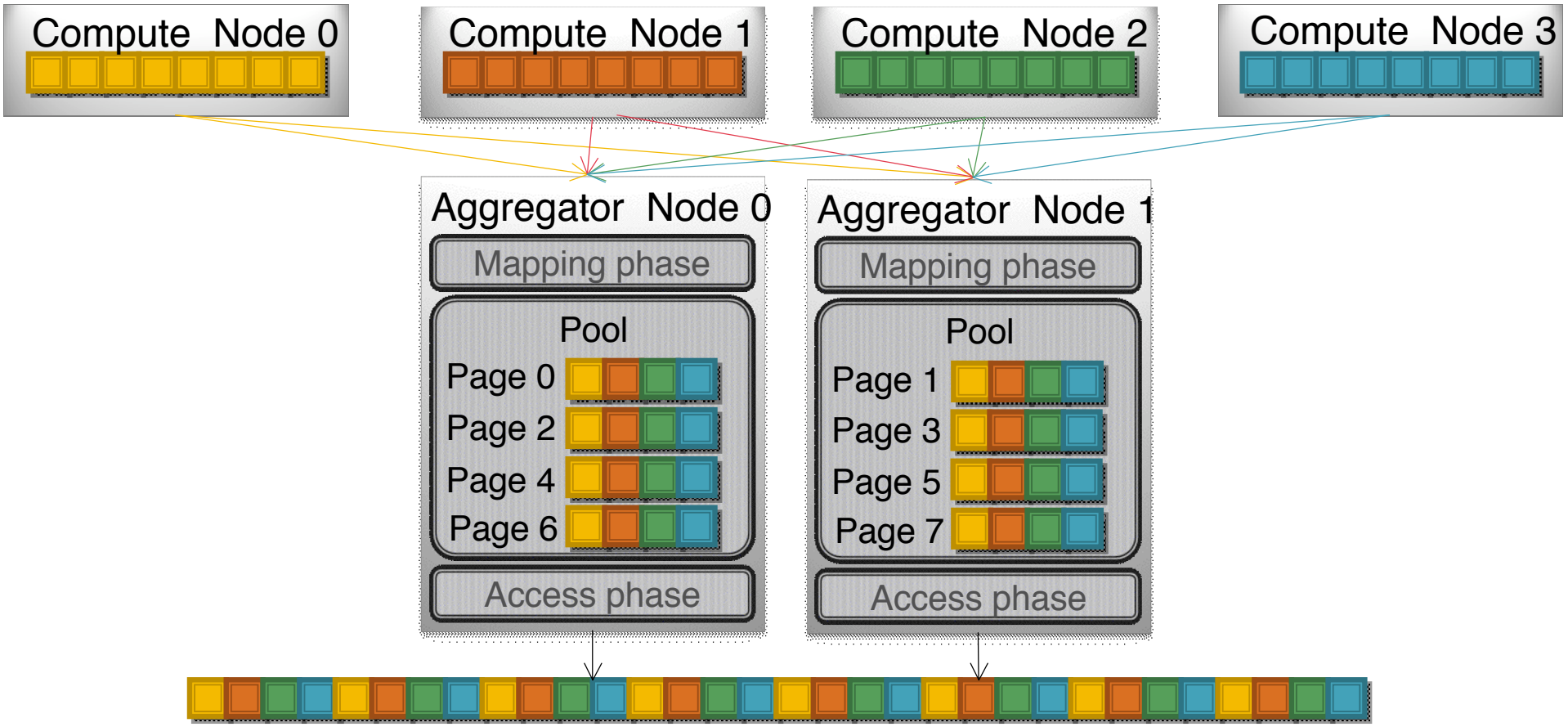
- We propose and evaluate an alternative to the two-phase collective I/O (TP I/O) implementation of ROMIO called view-based collective I/O (VB I/O).

- View based I/O targets the following goals:
  - **Reducing** the cost of data scatter-gather operations,
  - **Minimizing** the overhead of file metadata transfer,
  - **Decreasing** the number of conservative collective communication and synchronization operations.

View-based I/O

- Differences between two-phase I/O and view-based I/O :
  - At view declaration, **VB I/O** sends the view data type to aggregators, while **TP I/O** stores it locally at the application nodes.
  - **VB I/O** assigns statically the file domain to aggregators, while TP I/O dynamically.
  - At access time, **TP I/O** sends the offset-lists to the aggregators, while **view I/O** transfers only the view access interval extremities.
  - The collective buffers of **VB I/O** are cached across collective operations. A collective read following a write, may find the data already at the aggregator.
  - The collective buffers of **VB I/O** are written to the file system when the collective buffer pool is full or when the file is closed. For **TP I/O**, the collective buffers are flushed to the file system when they are full or at the end of each write operation.

Overview


Compute Node 0
Compute Node 1
Compute Node 2
Compute Node 3
Aggregator Node 0
Mapping phase
Pool
Page 0
Page 2
Page 4
Page 6
Access phase
Aggregator Node 1
Mapping phase
Pool
Page 1
Page 3
Page 5
Page 7
Access phase

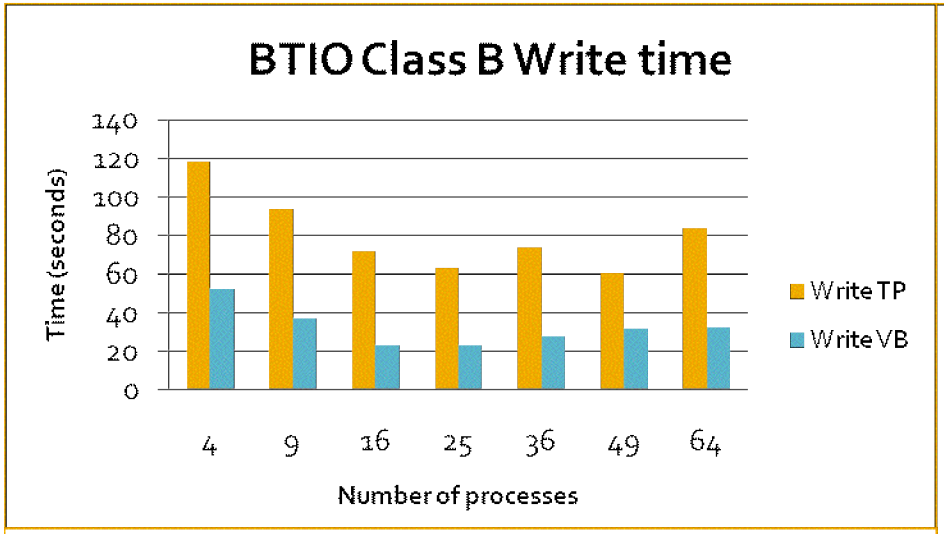
Evaluation

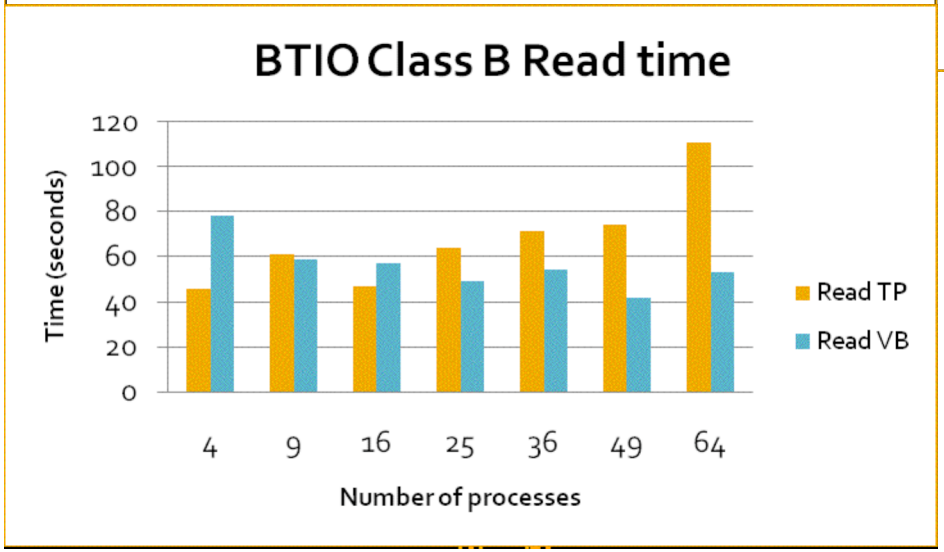
- Evaluated on CACAU (HLRS Stuttgart)
- MPICH2
- File system tested: PVFS 2.6.3 with 8 I/O servers
- The communication protocol of PVFS2 and MPICH2 was TCP/IP on top of the native Infiniband communication library
- 1 process per node
- View-based I/O had a collective buffer pool of maximum 64 Mbytes
- BTIO, coll perf and MPI_TILE_IO

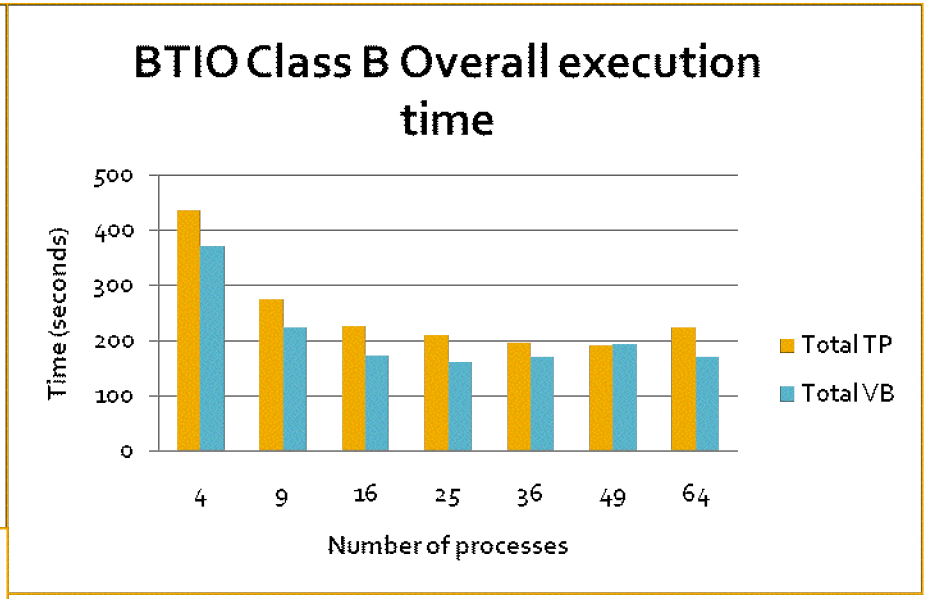
BTIO benchmark

- Use 4 to 64 processes and two classes of data set sizes: B (1697.93 Mbytes) and C (6802.44 MBytes).
- BTIO explicitly sets the size of write collective buffer to 1 Mbytes
- The benchmark reports the total time including the time spent to write the solution to the file.
- However, the verification phase time containing the reading of data from files is not included in the reported total time

BTIO benchmark



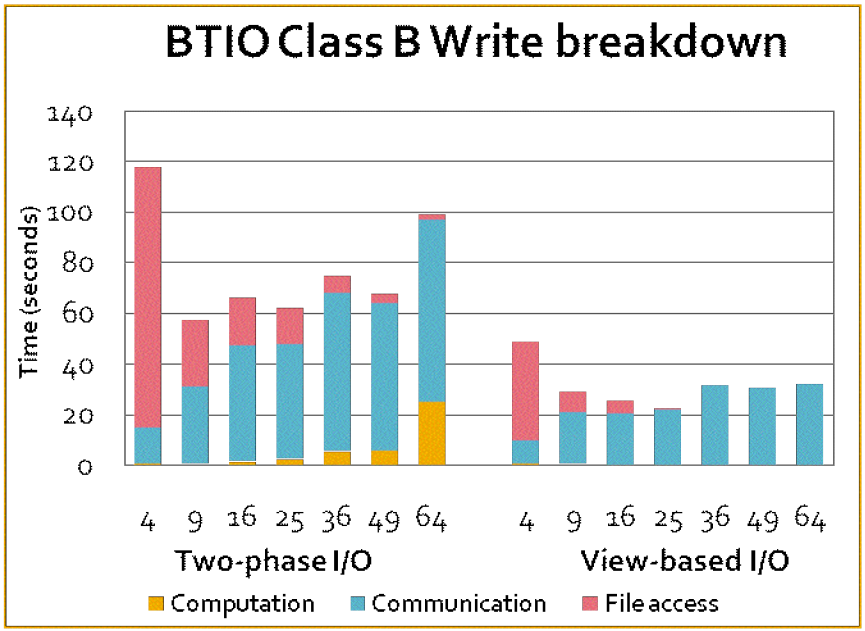


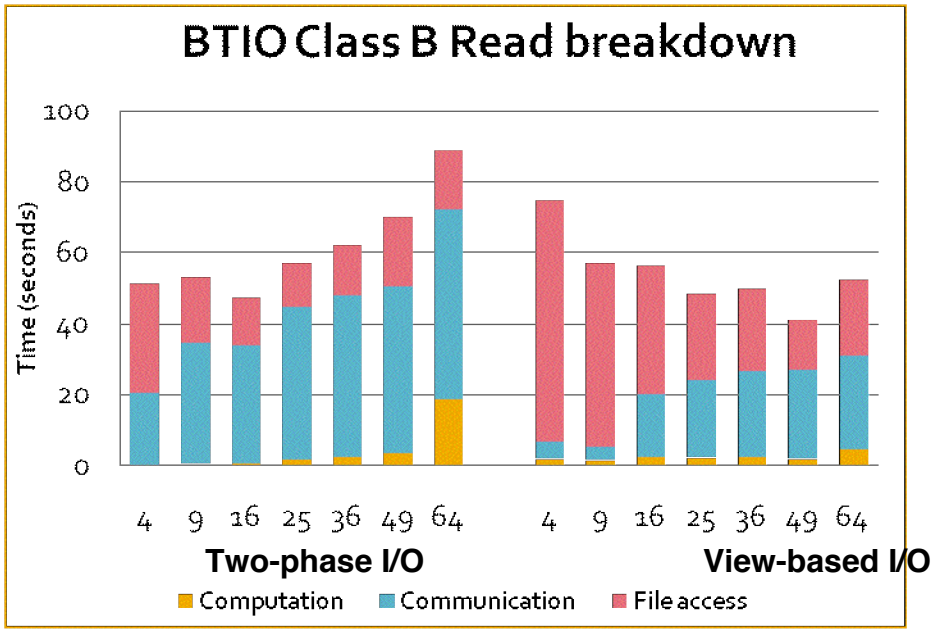


- Writes were between 89% and 121%
- Reads were between 3% to 109%
- Overral time was between 8% to 50%

BTIO breakdown

- Breakdowns: total time spent in computation, communication and file access of collective write and read operations, for class B from 4 to 64 processes.





Conclusions

- Avoids the necessity of transferring large lists of offset-length pairs at file access time as the present implementation of two-phase I/O.
- Reduces the total run time of a data intensive parallel application, by reducing both I/O cost and implicit synchronization cost.
- The write-on-close approach brings satisfactory results in all cases.

Future work

- Adding lazy view I/O
  - Views and data are sent together in write/read primitives
  - Views are sent if the aggregators do not have the data view
- Including two data staging strategies for **prefetching** and **flushing** the collective I/O buffer cache:
  - The prefetch is done in coordinate manner, by aggregating the view information of several processes and reading ahead whole blocks. Based on MPI-IO views.
  - The flushing strategy allows for overlapping the computation and I/O. Reduces also the rates at which the buffer cache becomes full with dirty file blocks, which may clog the computation to go on.
- Currently:
  - We have already implemented the mechanisms for enforcing these two strategies and are estimating the efficiency of this approach for large scale scientific parallel application.
  - We are investigating the trade-off between the contradictory goals of promoting data by prefetching, demoting the data by flushing and temporal locality.

Thanks for all