Filesystem Aware Scalable I/O Framework for Data-Intensive Parallel Applications

Rengan Xu[1] Mauricio Araya-Polo[2]
Barbara Chapman[1]

[1]Department of Computer Science
University of Houston

[2]Geophysics Development
Repsol

HPDIC Workshop 2013

Outline

Motivation

- Our targets are geophysical applications for oil and gas exploration
- Terabytes of seismic data is used both during I/O and computing. Many industrial applications face same challanges.
- Efficient use of parallel I/O within MPI applications
- Workers dont know the amount of data to write until runtime, and they work in embarrising parallel fashion, light synchronization during computing

Methodology

Task Scheduling Strategy

- Master: manage all tasks in global task queues
- Worker: request and execute tasks from global task queues

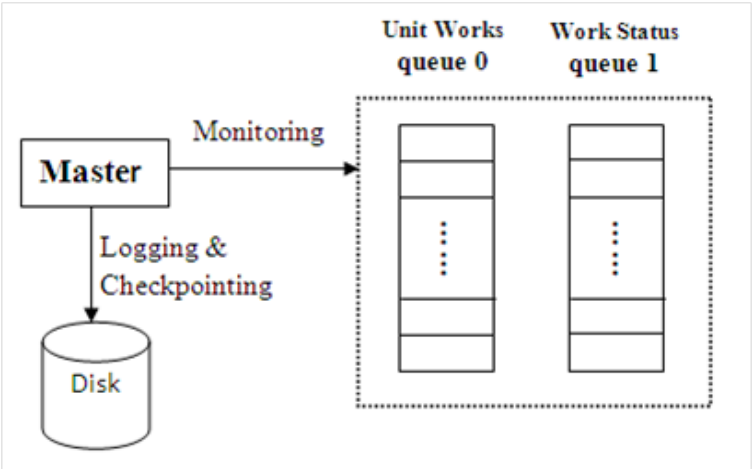

Figure: Dynamic Load Balancing Framework

Methodology

Parallel I/O Mechanism

- I/O node is added to coordinate communication
- An I/O FIFO mechanism is implemented
- The global I/O position is updated atomically
- The synchronization overhead among compute nodes is eliminated

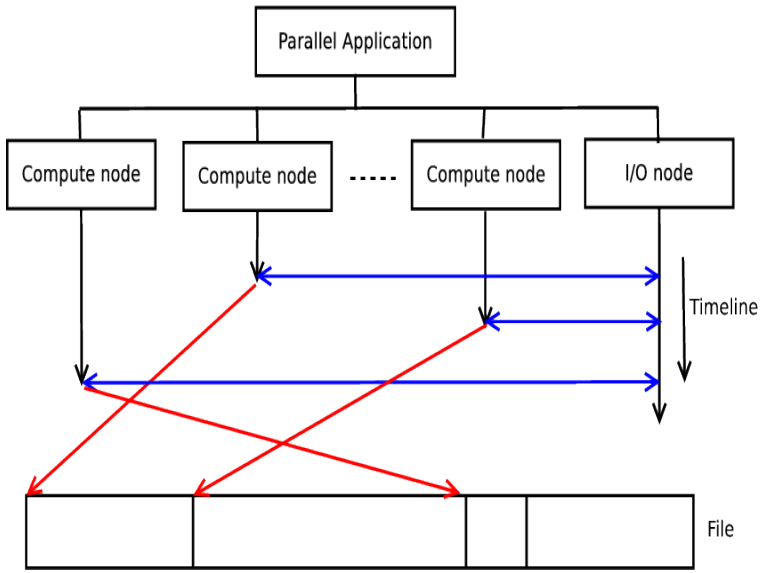

Figure: Parallel I/O Design in Target Applications

Methodology

Buffered I/O and Parallel I/O Interfaces

- The buffer size should be page-aligned and stripe-aligned
- The reason to be stripe-aligned is to avoid strip lock contention
- The buffer size that maximize the bandwidth is 4MB
- POSIX I/O uses lseek(), read() and write() to locate, read and write file.
- Memory-mapped file uses memory mapping to map a file from disk to process' address space
- Multiple processes can map the same file into memory to share data
- Page fault overhead when first loading file to memory

Methodology

Storage System Considerations

- Parallel file system: Panasas (v4.0.1)
- Clean data: data in client and storage are consistent
- Dirty data: data in client and the storage are inconsistent
- Read/Write (RW) and Concurrent Write (CW) mode (11.54 MB/s vs 147.86 MB/s)
- Potential contention even in CW mode, parallel writing serialized by **stripe lock**
- Solution: write data with multiple of parity stripe width size

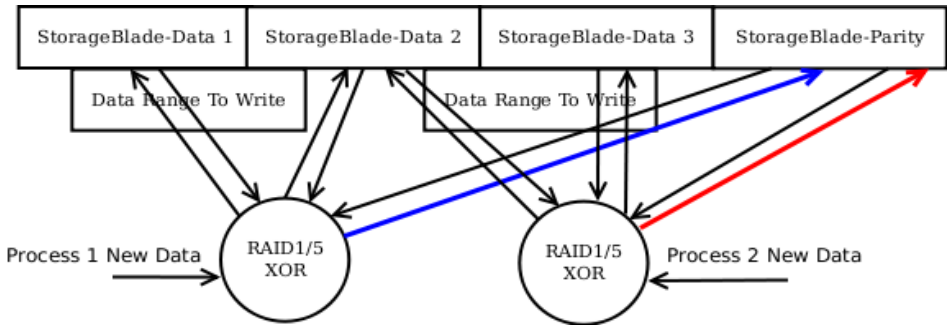

Figure: Stripe Lock Contention Example

Experimental Setup and Results

Experimental Setup

- Hardware: RAID 1/5
- Network between nodes: Infiniband (bandwidth is 40Gbit/s)
- Network connecting cluster and storage: 10 Gbit/s

Table: Configuration of each node in the cluster

Item	Description
Machine Type	x86_64
CPU Model	Intel Xeon X5675
CPU Cores	12 (6 x 2 sockets)
CPU Speed	3.07GHz
Memory Total	48G

Experimental Setup and Results

Results

- Output and input data file size: 100GB

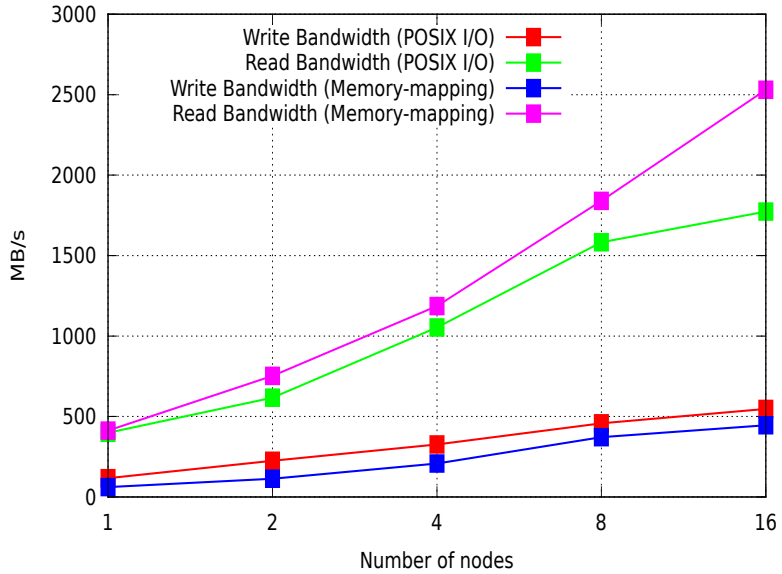

Figure: Bandwidth of Parallel I/O

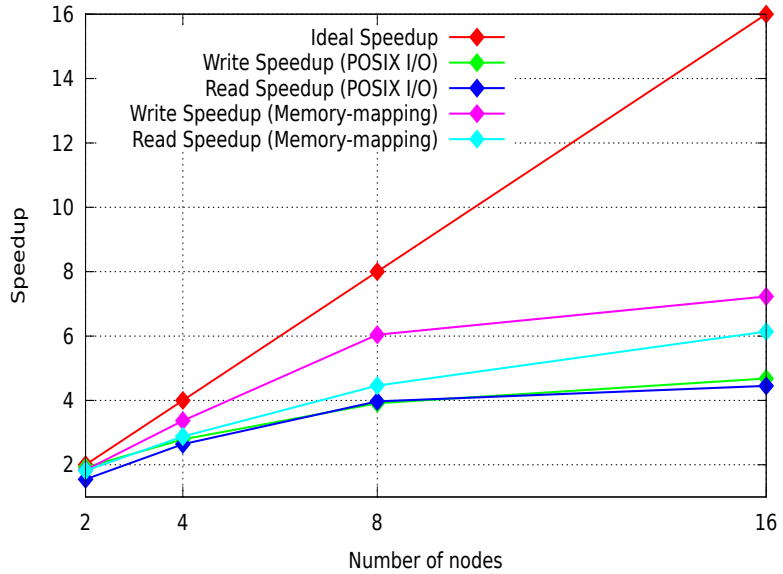

Figure: Speedup of Parallel I/O

Experimental Setup and Results

Write Time

- Serial[1] (disk-based strategy): every worker writes its output to the storage system, then the master reads on-by-one those outputs from the storage system and process them
- Serial[2] (network-based strategy): every worker sends its output to the master, which receive and process those outputs in serial fashion

I/O Approach	Nodes			
	1	4	8	16
Serial[1] (POSIX I/O)	**2007.19**	2777.92	3805.56	5860.84
Serial[2] (POSIX I/O)	**1770.28**	1830.28	1910.28	2070.28
Parallel (POSIX I/O)	**875.14**	313.35	223.94	187.06
Serial[1] (Memory-mapping)	**3574.04**	4318.88	5312.01	7298.24
Parallel (Memory-mapping)	**1662.88**	493.09	275.45	229.94

Experimental Setup and Results

Read Time

- Serial[1] (disk-based strategy): the master reads the input data and then send the data serially to each worker through the disk
- Serial[2] (network-based strategy): the master reads the input data and sends to each worker through the network

I/O Approach	Nodes			
	1	4	8	16
Serial[1] (POSIX I/O)	**1388.96**	4014.38	7514.94	14516.06
Serial[2] (POSIX I/O)	**276.91**	336.91	416.91	576.91
Parallel (POSIX I/O)	**256.91**	97.18	64.71	57.75
Serial[1] (Memory-mapping)	**2159.44**	7148.08	13799.60	27102.64
Parallel (Memory-mapping)	**248.28**	86.33	55.64	40.46

Conclusion and Future Work

Conclusion and Future Work

- Conclusion
  - Our solution reduces the global synchronization and communication overhead among all processes significantly
  - Ensure the dynamic load balancing, especially in a heterogeneous network
  - Our approach is independent of any parallel file system and hardware
  - Impressing bandwidth and speedup were achieved, overall scalability of target application can still be improved
  - Up to 30x write improvement and 250x improvement than the worst serial scenario, results using Panasas
- Future Work
  - Explore multiple I/O nodes if I/O requests is too intensive
  - Apply our strategy to finer granularity level: threads SMP

Load Balancing Profiling

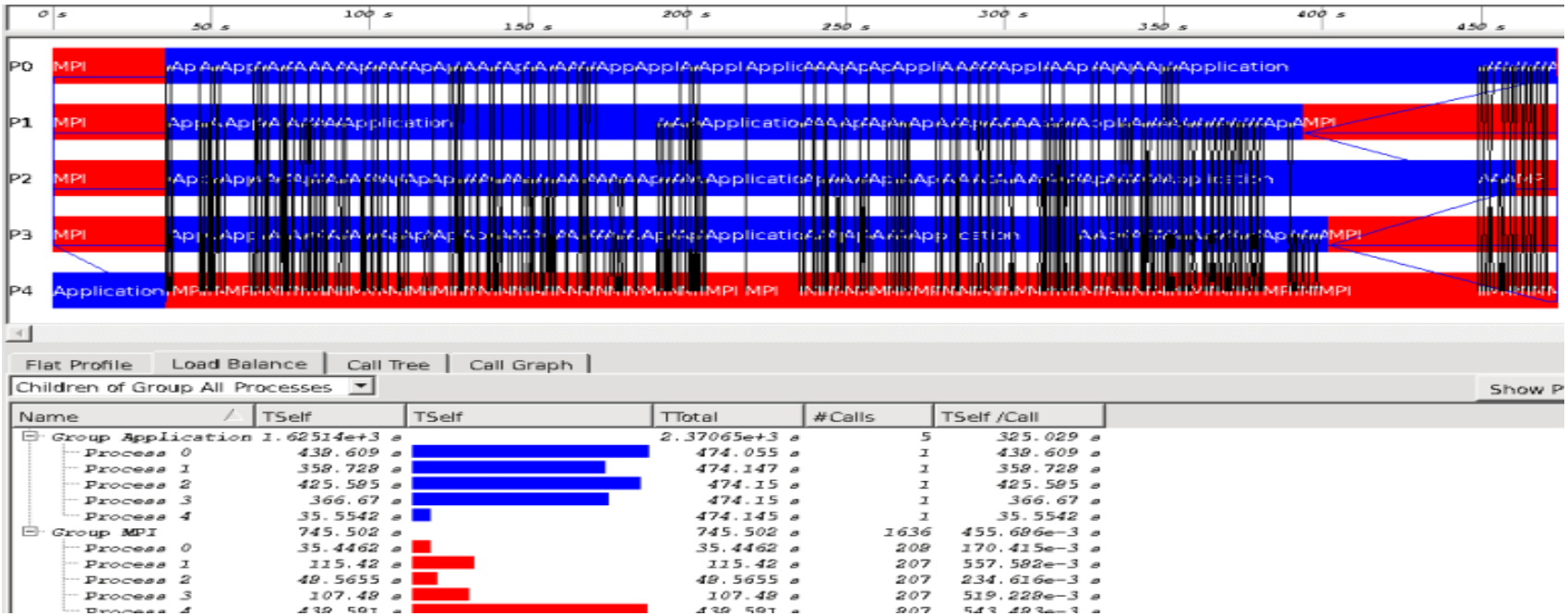

Figure: Load Unbalanced Profiling Result

Load Balancing Profiling

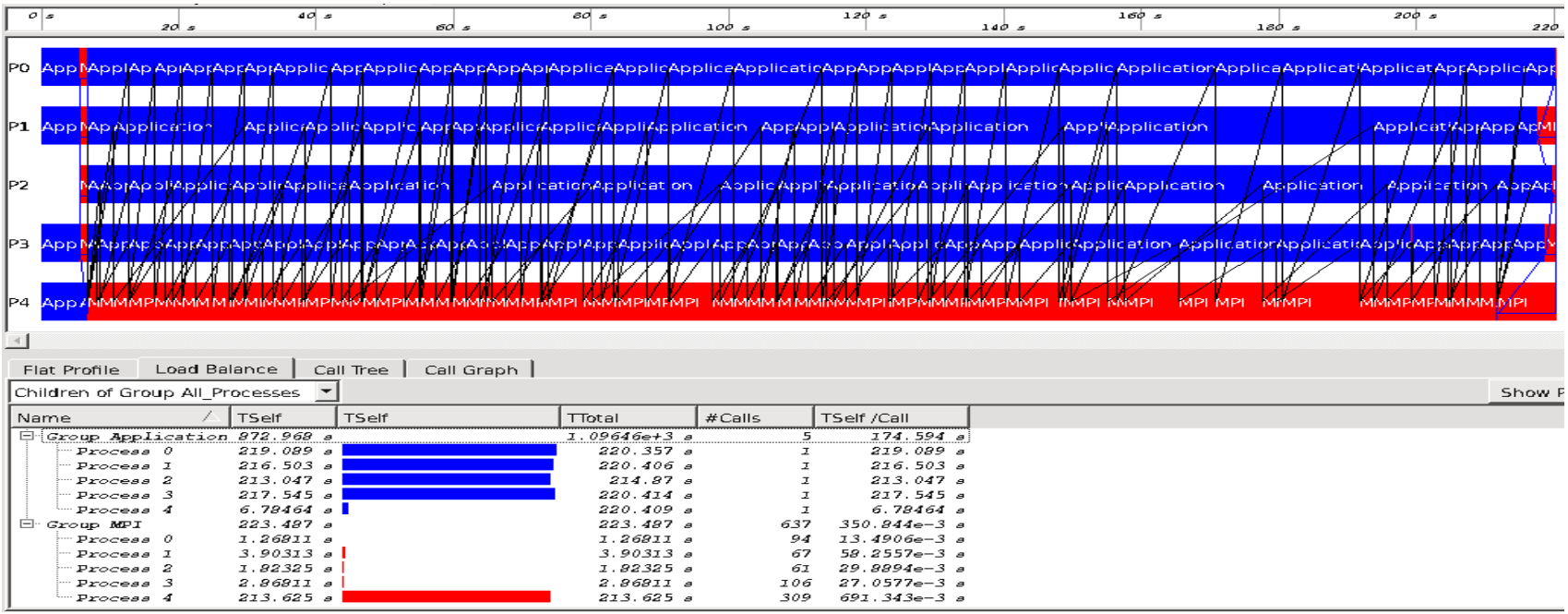

Figure: Load Balanced Profiling Result