Parallel I/O Framework for Data-Intensive Parallel Applications

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

[1]*University of Houston, USA;* [2]*Repsol, USA.*

Introduction

The motivation of this work are geophysical applications used for oil and gas exploration. These applications process Terabyte size datasets in HPC facilities. In general term, these applications read as inputs and write as intermediate/final results huge amount of data, where the underlying algorithms implement seismic imaging techniques. The traditional sequential I/O cannot complete all I/O operations for so large volumes of data in an acceptable time range. Even with parallel I/O, because of the dynamic property of many of these applications, each process does not know the data size it needs to write until its computation is done, and it also cannot identify the position in the file to write. In order to write correctly and efficiently, communication and synchronization are required among all processes to fully exploit the parallel I/O paradigm. Our approach removed the expensive synchronization and communication overhead effectively.

Methods

Task Scheduling Strategy The dynamic load balancing framework which uses Master/Worker model is applied.

- The master process is responsible to keep the workload balanced among all processes.
- The worker whose processing is faster will get more work, while those slow will get less work.
- It ensures good load balancing over heterogeneous nodes, especially the computational demands for each unit work are different.
- The master keeps monitoring the status of the work queue and puts their status into another queue. The status is saved into storage periodically

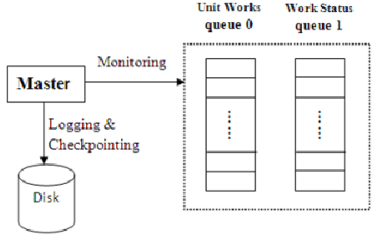

Figure 1: *Dynamic Load Balancing Framework*

Our Parallel I/O Approach

- In some of the target applications, each process does not know how much data it needs to write until the computation is finished.
- Our approach addes an I/O node that only handles I/O request.
- An I/O FIFO mechanism is implemented, where all compute nodes write data directly and in parallel.
- The synchronization among all compute nodes is eliminated and the only communication left is between compute nodes and I/O node.

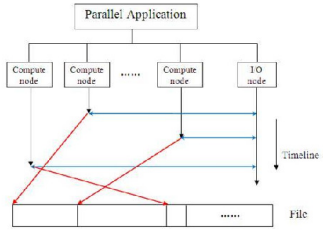

Figure 2: *Parallel I/O Design in Applications. Blue line indicates the communication between compute nodes and I/O node, and red line indicates writing data into file*

Buffered I/O The buffer size should be page-aligned (multiple of 4KB in our system) and stripe-align (multiple of parity stripe width size 512KB). Fig. 3 shows the bandwidth of different buffer sizes. Finally, 4MB is chosen since it has highest bandwidth.

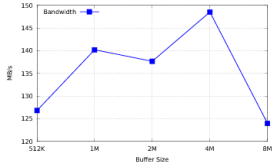

Figure 3: *Bandwidth with Different Buffer Size*

Storage System Considerations Our storage system is based on Panasas products. Although different compute nodes can write in parallel to storage, when two processes write close to each other, performance still may be degraded. Fig. 4 shows such an example.

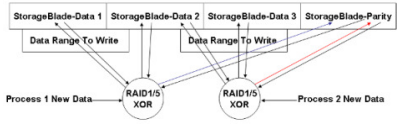

Figure 4: *Stripe Lock Contension Example. Process 1 and 2 try to update the same parity simultaneously while reading corresponding old data and parity and new data. To guarantee the correctness of new parity, parallel writing would be serialized internally by a "stripe lock". To avoid such lock contension, write should be stripe-aligned*

Results

Fig. 5 shows the bandwidth and speedup of parallel I/O experiments with both POSIX I/O and memory-mapping, respectively.

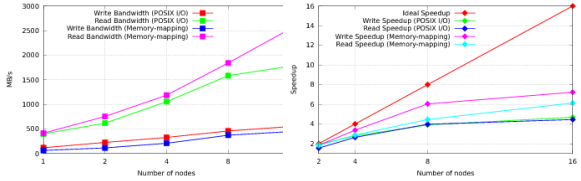

Figure 5: *Bandwidth and Speedup of Parallel I/O*

Table 1: *Elapsed time comparison between different approaches (time in seconds), dataset size 100 GB, only writing operations no read operations included*

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

Conclusions

- Our solution reduces the global synchronization and communication overhead among all processes significantly.
- With POSIX I/O, the speedup of parallel I/O is up to 4.68 in writing and 4.45 in reading with 16 processes.
- With memory-mapping, the speedup of parallel I/O is up to 7.23 in writing and 6.14 in reading with 16 processes.
- Our approach is independent of any parallel file system and hardware. Adapting it to other platforms will only require to set the proper parameters, such as stripe unit size, parity stripe width size, and buffer size.
- 30x improvement is achieved on the overall execution time of the application at hand, which greatly impact the projects turnaround where these applications are deployed.
- We used version 4.0.1 of Panasas client, more optimizations will be applied in newer client version, e.g. tuning of maximum readahead size, increase of receive buffer size on shelf in writing and on client in reading.

Acknowledgements: The author would like to thank Repsol for allowing us to present this work, and University of Houston for constant support.