Opportunistic Data-driven Execution of Parallel Programs for Efficient I/O Services

Xuechen Zhang
Song Jiang
Wayne State University

Kei Davis
Los Alamos National Laboratory
Process Scheduling in the HPC Systems

- Process scheduling is critical when computation is bottleneck
  - HPC systems achieve load balance among CPUs.
  - HPC systems reduce cache interference on the use of shared CPU caches.

- Process scheduler becomes less relevant when I/O is bottleneck
  - I/O time becomes longer than compute time.
  - Process scheduler is essentially in standby mode.
  - How to improve I/O efficiency by exploiting process scheduling?
Impact of Process Scheduling on Request Issuance Order

• Non-determinism of parallel program execution
  - The relative progress of each process is **nondeterministic**.
  - The order in which the processes issue their I/O requests is accordingly **nondeterministic** or **essentially random**.

• Study of process scheduling’s impact on I/O efficiency
  - Strategy 1 (Computation-driven method)
    - Conventional computation-driven execution
  - Strategy 2 (Application-level prefetching)
    - A prefetch request is issued **immediately** after it is generated.
  - Strategy 3 (Data-driven method)
    - Program processes are **suspended** and will be **resumed** only when the data to be requested are ready in memory.
    - The *pre-executing* threads generate a batch of I/O requests for efficient access from the disk to prepare the data.
Illustration of Program Execution Strategies

Strategy 1

| MPI Processes | Pre-execution Threads |

When I/O is bottleneck, I/O efficiency becomes the issue.
I/O Efficiency of Strategy 2

1) There are time gaps between requests issued during the pre-execution.
2) Requests can arrive at a data server in the random order.
3) Disk scheduler cannot see enough outstanding requests to explore spatial locality.
Illustration of Strategy 3

Significantly reduce I/O time because requests are sequentially served on the disks.
Strategy 3 increases execution time because MPI processes are suspended.

Strategy 3 speeds up execution by 36%.

1) A benchmark with stride data access pattern;
2) 10GB data file;
3) Eight processes with 4KB request size;
4) Eight data servers with 64KB striping unit size;
5) Various I/O ratios of I/O time over the program execution time.
On-disk Data Access Pattern

Strategy 3 reduces seek distance since it helps disk scheduler exploit I/O efficiency.
Data-driven Execution: Decoupling Process Scheduling from Request Issuance

- For the disk-based storage system, data-driven execution improves request spatial locality.
- MPI processes are suspended when I/O becomes the bottleneck and resumed to normal execution when CPU is the bottleneck.
- I/O requests are generated quickly by using pre-execution threads and served together with an improved efficiency.
Outline

• Motivation: Dual-mode Execution
• **Design and Implementation**
  • Evaluation
  • Related Work
  • Conclusions
DualPar: Dual-mode Execution of Parallel Programs

• Design objectives of DualPar
  - Enabling the new data-driven mode, in addition to existing computation-driven mode.
  - Adaptively and efficiently switching between the two modes according to current bottleneck.

• Challenges
  - How to determine current performance bottleneck?
  - How to predict data to be requested?
  - How to minimize penalty with mis-predictions?
When to Enter the Data-driven Execution Mode

- I/O intensities of programs are sufficiently large.
  - I/O ratio is larger than 80%.
- I/O efficiency of programs are low.
  - We use disk seek distance as the metric.
  - SeekDist represents the current I/O efficiency.
  - ReqDist represents the highest I/O efficiency that the system can achieve.
  - The ratio of SeekDist and ReqDist represents the potential I/O efficiency improvement.

\[
T_{\text{improvement}} = \frac{\text{avgSeekDist}}{\text{avg ReqDist}}
\]

Current Seeking Dist.  Shortest Seeking Dist.
Pre-execution for Predicting Future Requests

• Pre-execution of processes
  - Pre-execution threads are initialized as the MPI processes begin to issue I/O requests.
  - A buffer cache for each process is set up on its compute node.
  - Data service is either for the processes to write data to the cache or for the prefetcher to read data from the data servers into the cache.

• Overhead control
  - Miss-pretch ratio is defined as the fraction of prefetched but not used data in a cache when the next pre-execution begins.
  - When miss-pretch ratio is larger than a threshold, pre-execution is disabled.
Architecture of DualPar

Process Execution Control (PEC)

Compute Nodes

Execution Mode Control (EMC)

IO Ratio and Mis-prefetch Ratio

Disk Seek Distance

Locality

Disk I/O Scheduler

Hard Disk

Data Servers

Metadata server

Locality

Disk I/O Scheduler

Hard Disk
Execution Mode Transition

I/O ratio > 80% and Timprovement > 3

Computation–driven mode

Data–driven mode

I/O ratio < 80% or Timprovement < 3 or Mis-prefetch ratio > 20%.
Caching the Data for Computation

• Constructing a global cache
  ➢ To make efficient use of system memories, the caches must be shared among compute nodes to avoid redundancy and consistency issues.
  ➢ Memcached is used to provide a high-performance distributed cache infrastructure as an in-memory key-value store.

• Instrumenting MPI-IO library to use the global cache
  ➢ A file is partitioned into chunks of equal size.
  ➢ Every data chunk is indexed by a unique key generated from the name of the file that stores the chunk and chunk address in the file.
  ➢ Each chunk in the cache has a tag to record the time of its most recent reference. A chunk will be evicted if its is not used for a certain period of time.
A read miss on the cache will block the process and a pre-execution for the process will be initialized.

Write requests are directly served in the cache.

Mis-prefetch ratio is passed to the EMC daemon. Data-driven mode is disabled if the ratio is larger than a threshold.
Outline

• Motivation: Dual-mode Execution
• Design and Implementation
• Evaluation
• Related Work
• Conclusions
Experimental Setting

• Darwin cluster at Los Alamos National Laboratory
  ➢ 120 nodes and 8 data servers
  ➢ 48-core 2GHz AMD Opteron 6168, 64GB memory
  ➢ RAID0 consisting of two 500GB 7200rpm disks

• Software configuration
  ➢ Kernel 2.6.35.10
  ➢ CFQ for RAID
  ➢ MPICH2-1.4 compiled with ROMIO
  ➢ PVFS2 parallel file system
## Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Access Pattern</th>
<th>Sources</th>
</tr>
</thead>
<tbody>
<tr>
<td>mpi-io-test</td>
<td>contiguous data sets</td>
<td>PVFS2 software package</td>
</tr>
<tr>
<td>ior-mpi-io</td>
<td>non-contiguous data sets</td>
<td>the ASCI Purple benchmark suite developed at LLNL</td>
</tr>
<tr>
<td>noncontig</td>
<td>data access with vector-derived MPI data type</td>
<td>the Parallel I/O benchmarking Consortium at ANL</td>
</tr>
<tr>
<td>BTIO</td>
<td>small random</td>
<td>NAS Parallel benchmark from NASA</td>
</tr>
<tr>
<td>S3asim</td>
<td>random</td>
<td>Northwestern University and SNL</td>
</tr>
</tbody>
</table>
I/O throughput is increased by 105% and 57%, compared to vanilla and collective MPI-IO respectively. Due to unbalanced workload.
BTIO Benchmark

Throughput with collective IO and DualPar is larger than vanilla MPI-IO by up to 24X and 35X.

Performance advantage of collective IO gradually decreases. DualPar has better scalability.
iTransformer reduces I/O times by up to 25% and by 17% on average.
Heterogeneous Workloads

At first run `mpi-io-test` alone. `hpio` johns at fifth second.
Performance Impact of Cache Size

Throughput is increased by 43X.

Further increasing the cache size brings diminishing returns.
Related Work

• Data-driven process scheduling
  ➢ MapReduce [Dean et al. OSDI’04]
  ➢ DryadLINQ [Yu et al. SOSP’09]
  ➢ Dynamic sets [Steere SOSP’97]

• Pre-execution for I/O prefetching
  ➢ OS I/O speculation [Chang et al. OSDI’99,USENIX’03]
  ➢ Hiding parallel I/O time [Chen et al. SC’08]
Conclusions

• Data-driven execution mode is needed for improving I/O efficiency using thread pre-execution.

• An algorithm comprising of DualPar is proposed for execution mode adaptation to resource constraints.

• We implemented DualPar in the MPI-IO library, and experiments with both micro-benchmarks and macro-benchmarks show that DualPar can increase I/O throughput by 31% on average.
• Thank you!
## Overhead Analysis

<table>
<thead>
<tr>
<th>Cache Size (KB)</th>
<th>No DualPar</th>
<th>1024</th>
<th>2048</th>
<th>4096</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time (S)</td>
<td>138</td>
<td>140</td>
<td>142</td>
<td>148</td>
</tr>
</tbody>
</table>