Revealing Applications’ Access Pattern in Collective I/O for Cache Management

Yin Lu¹, Yong Chen¹, Rob Latham² and Yu Zhuang¹
Presented by Philip Roth³

¹Department of Computer Science
Texas Tech University
²Mathematics and Computer Science Division
Argonne National Laboratory
³Mathematics and Computer Science Division
Oak Ridge National Laboratory

28th International Conference on Supercomputing, 2014
1. Introduction
   - Background
   - Motivation

2. Collective I/O (CIO) Aware Cache Management
   - Overview
   - Design

3. Evaluation
   - Experimental Setup
   - Evaluation Results

4. Summary
Mountains of Data in Science

- Scientific applications, simulations, and visualizations produce and consume growing massive amounts of data
  - Sloan Digital Sky Survey (SDSS) generates 100TB data in each run
  - EarthScience accesses 3.5 PiB data within two months

(a) Molecular Science  (b) Climate Modeling  (c) Astrophysics
(d) Earth Science  (e) Health  (f) Life Science

Figure: Data Intensive Applications Extend Across a Wide Range of Science and Engineering Disciplines
Mitigating I/O bottlenecks

- Computing performance improvement rate magnitudes higher than that of I/O bandwidth
- Data access latencies further lag the system performance in exascale time
- Bridging the I/O chasm from different angles
  - Memory/Storage hierarchy
  - System software and libraries stack

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-level I/O Library</td>
</tr>
<tr>
<td>I/O Middleware (MPI-IO)</td>
</tr>
<tr>
<td>Parallel File System</td>
</tr>
<tr>
<td>I/O Hardware</td>
</tr>
</tbody>
</table>
Collective I/O services requests from all processes together

- Allowing the middleware take advantage of correlation between those requests
  - Filtering overlapping/redundant requests
  - Combining small and noncontiguous requests into large and contiguous ones

**Figure:** Collective I/O
Limitation of Current Collective I/O

- Detailed access patterns changed when the aggregated I/O request reaches the low level buffer cache

Figure: Collective I/O Hides Applications' Original Access Pattern Away
Limitation of Current Collective I/O

- Detailed access patterns changed when the aggregated I/O request reaches the low level buffer cache

Figure: Cache Layout Optimized with Proposed CIO-Aware Approach
Outline

1. Introduction
   - Background
   - Motivation

2. Collective I/O (CIO) Aware Cache Management
   - Overview
   - Design

3. Evaluation
   - Experimental Setup
   - Evaluation Results

4. Summary
High-Level View of Framework

Figure: Collective I/O Aware Cache Management Framework
CIO-aware Cache Management Scheme Overview

Pattern Detection Module
- Caching helper thread collects the original accesses
- Builds pattern values for caching library

Caching Library
- Maintains a global buffer cache among multiple processes
- Manages cache resource with a repertoire of replacement algorithms in the replacement module

Enhanced Collective I/O
- Checks the caching status of the requested blocks
- Fetches cached data from buffer cache directly
MPI I/O Access Pattern Detection

- Caching helper threads constructed when opening the file and destroyed when closing the file
- Pattern detection module receives local patterns and combines them into a global pattern
  - I/O operation type
  - spatial locality
  - temporal pattern
  - iterative behavior

Example: \{f[3], READ, 0.023184,1,[(2622716,510080),(1573632,510080)],64\}

Main Thread

```c
MPI_File_open (comm ,fname ,mode ,info ,& mpi_fh );
...
MPI_File_read_all (mpi_fh ,buf ,count ,dttype ,status );
...
MPI_File_close (& mpi_fh );
...```

Cache Helper Thread

```c
/* process rank */
rec->rank=thisrank;
/* file descriptor */
rec->filedes=mpi_fh->fd_sys;
...
/* individual file pointer */
rec->file_pos=mpi_fh->fp_ind;
...```
• Incorporating the caching into the MPI library
• Maintaining a global buffer cache in the client side
• Each client contributes part of its memory to construct the global cache pool
**Algorithm 1: CIO-Aware LRU**

*input*: A sequence of global pattern values $S_v$ from pattern detection module

*output*: The contents of buffer cache

1. **for each** global pattern value $g_v \in S_v$ **do**
   - split data requests with $g_v$ into blocks $B_s$;
   - uncaught data blocks set $U_s \leftarrow \emptyset$;
   - **for each** block $b_i \in B_s$ **do**
     - if $b_i \in$ buffer cache then
       - // cache hit
         - hits++;
         - // copy data $b_i$ to user using memcpy()
         - user specified buffer $\leftarrow b_i$ in buffer cache;
         - // update $b_i$ last access time
         - $Last(b_i) \leftarrow b_i$ time stamp;
     - else
       - // cache miss
         - // perform I/O from disk
         - user specified buffer $\leftarrow b_i$ in file system;
         - // evicting the LRU block
         - min $\leftarrow$ current time;
         - **for each** block $b_j \in$ buffer cache **do**
           - if $Last(b_j) <$ min then
             - victim $\leftarrow b_j$;
             - min $\leftarrow Last(b_j)$;
           - if victim $==$ dirty then
             - flush the victim to the disk;
             - fetch $b_i$ into the buffer frame held by victim;
             - $Last(b_i) \leftarrow b_i$ time stamp;

**Figure**: LRU with CIO Aware
CIO-Aware Cache Management

Figure: LFU with CIO Aware

Figure: ARC with CIO Aware
Experimental Settings

Platform

- 640-node Linux-based cluster test bed
- A 600TB Lustre file system and MPICH-3.0.2 library manage the storage system and runtime environment

Evaluation Metrics

- Two key metrics: application execution time (I/O throughput) and buffer cache hit rate
- Total execution time comparison each run with 10 GB data and the buffer cache size was set as 64MB per process
- Hit rate comparison each run with 32 processes
• Each process renders one tile with $1024 \times 1024$ pixels and the size of each element is 8 byte
• Total execution time comparison each run with 10 GB data and the buffer cache size was set as 64MB per process
The tests were carried out with 8MB I/O message size per process.
MPI-IO-Test Benchmark

(a) Total Execution Time Comparison

(b) Hit Rate Comparison
Outline

1. Introduction
   - Background
   - Motivation

2. Collective I/O (CIO) Aware Cache Management
   - Overview
   - Design

3. Evaluation
   - Experimental Setup
   - Evaluation Results

4. Summary
• Collective I/O leverages correlations among parallel processes to carry out parallel I/O requests more efficiently
• The current collective I/O filters away useful access patterns during the aggregation process
• Proposed collective I/O aware cache management methodology reveals unseen access pattern to underlying caching algorithms
• Initial evaluation results are promising
• Hope can be helpful and can provide guidance to build an even more efficient parallel I/O system
Thank You

For more information, please visit:
http://discl.cs.ttu.edu