Do You Know What Your I/O Is Doing?

William Gropp

www.cs.illinois.edu/~wgropp
Messages

• Current I/O performance is poor
  ♦ Even relative to what current systems can achieve
  ♦ Part of the problem is the I/O interface semantics

• Big data is more than just I/O
  ♦ HPC has relevant insights
Just How Bad Is Current I/O Performance?

• Much of the data (and some slides) taken from “A Multiplatform Study of I/O Behavior on Petascale Supercomputers,” Huong Luu, Marianne Winslett, William Gropp, Robert Ross, Philip Carns, Kevin Harms, Prabhat, Suren Byna, and Yushu Yao, presented at HPDC’15.
  ♦ This paper has lots more data – consider this presentation a sampling

• Thanks to Luu, Behzad, and the Blue Waters staff and project for Blue Waters results
  ♦ Analysis part of PAID program at Blue Waters
I/O Logs Captured By Darshan, A Lightweight I/O Characterization Tool

- Instruments I/O functions at multiple levels
- Reports key I/O characteristics
- Does not capture text I/O functions
- Low overhead \(\rightarrow\) Automatically deployed on multiple platforms.
Caveats on Darshan Data

• Users can opt out
  ♦ Not all applications recorded; typically about ½ on DOE systems

• Data saved at MPI_Finalize
  ♦ Applications that don’t call MPI_Finalize, e.g., run until time is expired and then restart from the last checkpoint, aren’t covered

• About ½ of Blue Waters Darshan data not included in analysis
  ♦ Expect to be fixed soon
I/O log dataset: 4 platforms, >1M jobs, almost 7 years combined

<table>
<thead>
<tr>
<th></th>
<th>Intrepid</th>
<th>Mira</th>
<th>Edison</th>
<th>Blue Waters</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecture</td>
<td>BG/P</td>
<td>BG/Q</td>
<td>Cray XC30</td>
<td>Cray XE6/ XK7</td>
</tr>
<tr>
<td>Peak Flops</td>
<td>0.557 PF</td>
<td>10 PF</td>
<td>2.57 PF</td>
<td>13.34 PF</td>
</tr>
<tr>
<td>Cores</td>
<td>160K</td>
<td>768K</td>
<td>130K</td>
<td>792K+59K smx</td>
</tr>
<tr>
<td>Total Storage</td>
<td>6 PB</td>
<td>24 PB</td>
<td>7.56 PB</td>
<td>26.4 PB</td>
</tr>
<tr>
<td>Peak I/O Throughput</td>
<td>88 GB/s</td>
<td>240 GB/s</td>
<td>168 GB/s</td>
<td>963 GB/s</td>
</tr>
<tr>
<td>File System</td>
<td>GPFS</td>
<td>GPFS</td>
<td>Lustre</td>
<td>Lustre</td>
</tr>
<tr>
<td># of jobs</td>
<td>239K</td>
<td>137K</td>
<td>703K</td>
<td>300K</td>
</tr>
<tr>
<td>Time period</td>
<td>4 years</td>
<td>18 months</td>
<td>9 months</td>
<td>6 months</td>
</tr>
</tbody>
</table>
Very Low I/O Throughput Is The Norm
Most jobs transfer little data. Many big-data jobs also have very low throughput.
Most Jobs Read/Write Little Data (Blue Waters data)
I/O Thruput vs Relative Peak

Number of processes

I/O Throughput

1TB/s
1 GB/s
1 MB/s
1 KB/s

5.35 GB/s per 125 nodes

Jobs Count
1 - 10
11 - 100
101 - 500
501 - 1k
1k1 - 5k
5k1 - 10k
10k1 - 50k

Iols
~50% of apps never transfer > 1GB

~20% of apps use only text I/O
I/O Time Usage Is Dominated By A Small Number Of Jobs/Apps
Improving the performance of the top 15 apps can save a lot of I/O time

<table>
<thead>
<tr>
<th>Platform I/O time percent</th>
<th>Percent of platform I/O time saved if min throughput = 1 GB/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mira</td>
<td>83%</td>
</tr>
<tr>
<td>Intrepid</td>
<td>73%</td>
</tr>
<tr>
<td>Edison</td>
<td>70%</td>
</tr>
<tr>
<td>Blue Waters</td>
<td>75%</td>
</tr>
</tbody>
</table>
Top 15 apps with largest I/O time (Blue Waters)

- Consumed 1500 hours of I/O time (75% total system I/O time)
POSIX I/O is far more widely used than parallel I/O libraries.
What Are Some of the Problems?

• POSIX I/O has a strong consistency model
  ♦ Hard to cache effectively
  ♦ Applications need to transfer block-aligned and sized data to achieve performance

• Files as I/O objects add metadata “choke points”
  ♦ Serialize operations, even with “independent” files

• Burst buffers will *not* fix these problems – must change the semantics of the operations

• “Big Data” file systems have very different consistency models and meta data structures, designed for their application needs
  ♦ Why doesn’t HPC?

• There have been some efforts, such as PVFS, but the *requirement* for POSIX has held up progress
Big Data is More Than I/O

• One example is distributed, out-of-core graph processing
  ♦ Constantly growing graph sizes with large memory footprints
  ♦ Current distributed graph processing frameworks assume graphs fit in memory
    • Including all intermediate states
    • “Easy” but expensive fix is very large memory nodes
  ♦ Can we use out-of-core techniques?

• This is work of Hassan Eslami, conducted during a summer internship at Facebook
Solution

• We need a strategy to automatically and intelligently decide which data should be in-memory or out-of-core.

• This is done by:
  - Adaptive control of in-memory data
  - Congestion control of incoming messages
  - Capacity control of outgoing messages
Adaptive Control of In-memory Data

- **Data usage > High:** offload data to disk until usage below Mid
- **Data usage < Low:** lazily load data of latest offload from disk
Congestion Control of Incoming Messages

Worker → Disk

Worker → Worker

Worker → Worker

Worker → Worker
Congestion Control of Incoming Messages
Capacity Control of Outgoing Messages

- Keeps a count of outgoing on-the-fly messages per worker pair
- Limits in-transit messages per each worker pair in a two phase approach
  1. $\text{count} > \text{MAX-IN-TRANSIT}$: cache the message
  2. $\text{size(cache)} > \text{MAX-CACHE-SIZE}$: stop computation
Result

- Implementation now available in Apache Giraph
- Results for PageRank on 8 workers on an input graph where graph data and messages take roughly 650GB with CMS as garbage collection strategy

![Graph showing execution time and available memory in each worker.](image)
Observations

• Dealing with large graphs requires fast messaging
  ♦ Issues such as memory management of “eager” data, flow control, nonblocking operations are important
  ♦ Latency hiding in I/O also important

• Common programming model is BSP
Message 1

• Current I/O performance is poor
  ♦ Metadata operations often a significant source of poor performance
  ♦ Related to mismatch between system and user expectations

• CS Challenge: Better I/O consistency and programming models
• Math Challenge: Match algorithms to realities of (changing) hardware; need aggregates, realistic model of data transfer costs
Message 2

- Big data is more than just I/O
  - And more than just operations on nearly independent data, for example...
  - Need to handle large graphs
  - CS Challenge: Low latency, high bandwidth, latency hiding programming and implementation, including multiple levels of memory hierarchy
  - Math Challenge: Match algorithms to problems; exploit years of effective sparse matrix work
Thanks!

- Especially Huong Luu, Babak Behzad, Hassan Eslami
- Funding from:
  - NSF
  - Blue Waters
- Partners at ANL, LBNL; DOE funding
- Internship support for Eslami from Facebook