SpongeFiles:
Mitigating Data Skew in MapReduce Using Distributed Memory

Khaled Elmeleegy
Turn Inc.
kelmeleegy@turn.com

Christopher Olston
Google Inc.
olston@google.com

Benjamin Reed
Facebook Inc.
br33d@fb.com
Background

• MapReduce are the primary platform for processing web & social networking data sets

• These data sets tend to be heavily skewed
  • Native : hot news, hot people(e.g. holistic aggregation)
  • Machine learning: “unknown topic” “unknown city”

• Skew leads to overwhelm that node's memory capacity
Data Skew: *harmness & solution*

• Harmness
  • The *major slow down* of the MR job

• Solution
  • Provide sufficient memory
  • Use data skew avoidance techniques
    • ......
Solution 1: Provide Sufficient Memory

• Method:
  • Providing every task with enough memory

• Shortcoming:
  • Tasks are only run-time known
  • Required memory may not exist at the node executing the task

• Conclusion:
  • Although can mitigate spilling, but very wasteful
Solution2: *Data skew avoidance techniques*

- **Method:**
  - *Skew-resistant partitioning* schemas
  - *Skew detection and work migration* techniques

- **Shortcoming:**
  - UDFs may be vulnerable to data skew

- **Conclusion:**
  - Alleviating some, but not all
Sometimes, we have to resort to Spill
Hadoop’s Map Phase Spill

By default 100MB

kvindeces[] 3.75%
kvoffset[] 1.25%
kvbuffer[] 95%
Hadoop’s Reduce Phase Spill

- Memory buffer
- LocalDisk
- LocalFSMerger
- InMemFSMergerThread
- merge
- Spill to Disk
- In Memory
- On Disk
- MapOutputCopier
- copiers
How we expect that we can share the memory ......
Here comes the SpongeFile

Share memory in the same node

Share memory between peers
SpongeFile

• Utilize remote idle memory
• Operates at app level (unlike remote memory)
• Single spilled obj stored in a single sponge file
• Composed of large chunks
• Be used to complement a process's memory pool
• Much simpler than regular files (for fast read & write)
Design

• No concurrent
  • single writer and a single
• Does not persist after it is read
  • lifetime is well defined
• Do not need a naming service
• Each chunk can lie in the :
  • machine's local memory,
  • a remote machine's memory,
  • a local file system, or a distributed file system
Local Memory Chunk Allocator

**Effect:**
Share memory between tasks in the same node

**Steps:**
1. Acquires the shared pool's lock
2. Tries to find a free chunk
3. Release the lock & return the chunk handle (meta data)
4. Return a error if no free chunk
Remote Memory Chunk Allocator

**Effect:**
Share memory between tasks among peers

**Steps:**
1. Get the list of sponge servers with free memory
2. Find a server with free space *(on the same rack)*
3. Writes data & gets back a handle
Disk Chunk Allocator

Effect:

It is the last resort, similar to spill on disk

Steps:

1. Tries on the underlying local file system
2. If local disks have no free space, then tries the distributed file systems
Garbage Collection

Tasks are alive:
delete their SpongeFiles before they exit

Tasks failed:
sponge servers perform periodic garbage collections
Potential weakness analytics

• May Increase the probability of task failure
• But:

\[ P = 1 - \left( e^{-\frac{N \cdot t}{MTTF}} \right) \]

= 1% per month

• N : number of machine
• t : running time
• MTTF : mean time to failure
Evaluation

• Microbenchmarks
  • 2.5Ghz quad core Xeon CPUs
  • 16GB of memory
  • 7200RPM 300G ATA drives
  • 1Gb Ethernet
  • Red Hat Enterprise Linux Server release 5.3
  • Ext4 fs

• Macrobenchmarks
  • Hadoop 0.20.2 of 30 node(2 map task slots & 1 reduce)
  • Pig 0.7
  • With above
# Microbenchmarks

Spill a 1 MB buffer 10,000 times to disk and mem

<table>
<thead>
<tr>
<th>In Memory</th>
<th>Time(ms)</th>
<th>On Disk</th>
<th>Time(ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Local shared memory</strong></td>
<td>1</td>
<td><strong>Disk</strong></td>
<td>25</td>
</tr>
<tr>
<td><strong>Local memory</strong> (through sponge server)</td>
<td>7</td>
<td><strong>Disk with background IO</strong></td>
<td>174</td>
</tr>
<tr>
<td><strong>Remote memory (over the network)</strong></td>
<td>9</td>
<td><strong>Disk with background IO and memory pressure</strong></td>
<td>499</td>
</tr>
</tbody>
</table>
Microbenchmarks’s conclusion

1. Spilling to shared memory is the least expensive

2. Then comes spilling locally via the local sponge (more processing and multiple message exchanges)

3. Disk spilling is two orders of magnitude slower than memory
Macrobenchmarks

• The jobs’ data sets:

<table>
<thead>
<tr>
<th></th>
<th>Input Bytes</th>
<th>Spilled Bytes</th>
<th>Spilled Chunks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Median Frequent Anchortext Spam Quantiles</td>
<td>10 GB</td>
<td>10.3 GB</td>
<td>10527</td>
</tr>
<tr>
<td></td>
<td>2.5 GB</td>
<td>7.2 GB</td>
<td>7383</td>
</tr>
<tr>
<td></td>
<td>3 GB</td>
<td>10.2 GB</td>
<td>10478</td>
</tr>
</tbody>
</table>

• Two versions of Hadoop:
  • The original
  • With SpongeFile

• Two configuration of memory size:
  • 4GB
  • 16GB
• When memory size is small, spilling to SpongeFiles performs better than spilling to disk

• When memory is abundant, performance depends on the amount of data spilled and the time difference between when the data is spilled and when it is read back
Using SpongeFiles reduces the job1’s runtime by over 85% in case of disk contention and memory pressure (similar behavior is seen for the spam quantiles).

For the frequent anchor text job, when memory is abundant and even with disk contention, spilling to disk performs slightly better than spilling to SpongeFiles.
• No spilling performs best
• Spilling to local sponge memory performs second
• But spilling to SpongeFiles is the only one practical
Related work

• Cooperative caching (for share)
• Network memory (for small objects)
• Remote paging systems (not the same level)

Conclusion

• Complementary to skew avoidance
• Reduce job runtimes by up to 55% in absence of disk contention and by up to 85%