Evaluation of Codes with Inherent Double Replication for Hadoop

M. Nikhil Krishnan,
N. Prakash,
V. Lalitha,
Birenjith Sasidharan,
P. Vijay Kumar

Srinivasan Narayanamurthy,
Ranjit Kumar,
Siddhartha Nandi
Acknowledgement:

Many thanks go out to USENIX for the generous Travel Grant that made this trip possible!!!
An Overview
Double Versus Triple Replication

- Triple replication of data is common in Hadoop:
  - Data availability for MapReduce (MR) operations
  - Resiliency to node failure
  - Storage overhead (OH)

- Double replication in comparison:
  - Data availability for MR operations (moderate workloads)
  - Resiliency to node failure
    - Via additional parities (checksums)
    - Eg: RAID + mirroring (RAID + m)
  - Reduced OH
• A simple example of double replication with added parity

• P is an XOR of the 9 data blocks

• The two parity blocks ensure adequate resiliency
The Heptagon-Local Code (HLC) : An Alternate Code with Inherent Double Replication

Looks complicated, but is not as we shall see..

### Performance Comparison:

**Overhead (OH) and Resiliency (MTTDL)**

<table>
<thead>
<tr>
<th>Code</th>
<th>OH</th>
<th>MTTDL (in years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-REP</td>
<td>2</td>
<td>1.5E+05</td>
</tr>
<tr>
<td>3-REP</td>
<td>3</td>
<td>1.2E+09</td>
</tr>
<tr>
<td>RAID + m</td>
<td>2.22</td>
<td>2.0E+09</td>
</tr>
<tr>
<td>HLC</td>
<td>2.15</td>
<td>8.3E+09</td>
</tr>
</tbody>
</table>

The HLC has reduced overhead for the desired resiliency but there is an issue relating to **Data Locality**....
Map Task Assignment: Local vs Remote Tasks

DATA 1
Free Processor
Free map slot

DATA 2
Busy Processor
Busy map slot

DATA n

Node 1
Local task

Node 2

Node 3
Remote task

Node 4
Importance of Data Locality for Map Tasks

- Data Locality measured by percentage of tasks that are executed locally


The Issue with Data Locality

- The HLC does not fare well in comparison with RAID+m in terms of locality for Map Task Assignment

- We will shortly see the reason for this

- One work around is to have a larger number of processors per node as we do here.
An Alternate Idea of Dealing with the Data Locality Issue

• Modify the HDFS to permit coding across files – this would eliminate this issue altogether (but not part of the present work)

• Other options?

.......future work!!!
And now for the details....
Explaining the Origins of the Heptagon-Local Code (HLC)
Regenerating Codes and Locally Repairable Codes are two classes of codes designed with distributed storage in mind.
The Pentagon Code

- An example of a code with inherent double replication
- 9 data blocks encoded into 20 blocks and stored in 5 nodes
- Each node stores 4 blocks
Building the Pentagon Code

- Input: 9 data blocks

Step 1: Adding a Single Checksum

- Input: 9 data blocks
- An additional XOR parity is added
Step 2: Label the Edges of the Graph with these 10 Blocks

- Input: 9 data blocks
- An additional XOR parity is added
- The resultant 10 symbols are placed on the edges of a fully-connected pentagon
Each node stores the data appearing on its incoming edges and we are done!

Input: 9 data blocks

An additional XOR parity is added

The resultant 10 symbols are placed on the edges of a fully-connected pentagon

Each node stores the data appearing on its incoming edges and we are done!
The Pentagon Code can be viewed as a compact rearrangement of the block of the (10,9) RAID+m code.
Resiliency of the Pentagon Code

- Can tolerate 2 (out of 5) node failures
Resiliency of the Pentagon Code

- Can tolerate 2 (out of 5) node failures
- All blocks except with the exception of one block can be copied from the other 3 nodes
Resiliency of the Pentagon Code

- Can tolerate 2 (out of 5) node failures

- All blocks except with the exception of one block can be copied from the other 3 nodes

- Block shared between the two failed nodes, recovered via parity
Extending the Pentagon to a Heptagon

- Construction extends to a polygon of any size

Pentagon: (10, 9) RAID + m

Heptagon: (21, 20) RAID + m
**Extending the Pentagon to a Heptagon**

- Construction extends to a polygon of any size

Pentagon: (10, 9) RAID + m

Heptagon: (21, 20) RAID + m

OH decreases
\[
\frac{42}{20} < \frac{20}{9}
\]

But resiliency is reduced
\[
\frac{2}{7} < \frac{2}{5}
\]
Finally: The Heptagon-Local Code (HLC)

Global parities offer a simple way to increase the resiliency of the Heptagon Code, at a slight increase in OH

• Overall: 40 data blocks → 86 encoded blocks
• Stored in 15 nodes
• Can tolerate any 3 (out of 15) node failures
Heptagon-Local Code: Recovering from Two Node Erasures

2 node failures within a Heptagon recovered locally (as in the Pentagon)
Heptagon-Local Code: Recovering from Three Node Erasures

3 node failures within a Heptagon recovered using the two parities and with the help of the second Heptagon
Overhead vs Resiliency

<table>
<thead>
<tr>
<th>Code</th>
<th>Overhead</th>
<th>MTTDL (in years)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2-REP</td>
<td>2</td>
<td>1.46E+05</td>
</tr>
<tr>
<td>3-REP</td>
<td>3</td>
<td>1.20E+09</td>
</tr>
<tr>
<td>Pentagon</td>
<td>2.22</td>
<td>1.05E+08</td>
</tr>
<tr>
<td>Heptagon</td>
<td>2.1</td>
<td>2.68E+07</td>
</tr>
<tr>
<td>(10, 9) RAID + m</td>
<td>2.22</td>
<td>2.03E+09</td>
</tr>
<tr>
<td>(12,11) RAID + m</td>
<td>2.18</td>
<td>6.50E+08</td>
</tr>
<tr>
<td>Heptagon-Local</td>
<td>2.15</td>
<td>8.34E+09</td>
</tr>
</tbody>
</table>

- MTTDL calculated for a 25 node system using standard models
- The heptagon-local code is a better choice than RAID + m

Returning to the Data Locality Issue
Data Locality for the Heptagon Code

- Data Locality affected by the concentration of data blocks in the Heptagon Code

- Say 2 map slots / node and MR job on 1 Heptagon-Coded file
  - No. of local tasks $\leq (7 \times 2) = 14$
  - No. of remote tasks $\geq (20 - 14) = 6$

- Data Locality: $\text{RAID} + m > \text{Heptagon-Local}$
Evaluation of MR Performance in Hadoop
Experimental Set-up

- Implemented the Pentagon and Heptagon-Local Codes on top of HDFS-RAID
- HDFS-RAID: Facebook's open-source implementation of Reed-Solomon codes over HDFS

<table>
<thead>
<tr>
<th></th>
<th>SET-UP A</th>
<th>SET-UP B</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processors (Map Slots)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Per Node</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nodes</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>Platforms</td>
<td>Dual-Core Laptops</td>
<td>Server-Class Machines</td>
</tr>
<tr>
<td>Codes Tested</td>
<td>Pentagon, Heptagon-Local*</td>
<td>Pentagon</td>
</tr>
</tbody>
</table>

* The Heptagon code was used in the experiments, but both Heptagon and Heptagon-Local have the same Map-Reduce performance
MR Performance (Batch Jobs)

• As expected, substantial loss in performance with 2 map slots.

• With 4 map slots, pentagon code performs well, even at load of 75%.
Summary

• We discussed a coding scheme – the Heptagon-Local Code, having inherent double replication for data blocks

• The Heptagon-Local Code enjoys good overhead and resiliency, when compared with other schemes such as RAID +m

• For good data locality, a larger number of processor cores / node is needed
Future Work: Restoring Data Locality

- Modify the HDFS to permit erasure coding across files which would eliminate the issue of data concentration.
Related Work: Erasure Codes in Distributed Storage

- Implementation of locally repairable codes, focus on overhead vs repair-cost
  (Huang et al, Sathiamoorthy et al)

- Regenerating codes for cloud based systems, focus on decreasing repair bandwidth
  (Hu et al, Runhui Li et al)

- A new erasure coding scheme having better single-node repair cost than a Reed-Solomon code
  (Rashmi et al)

- A new class of codes (rotated Reed-Solomon codes) for improved degraded read performance
  (Khan et al)

These codes possess a single copy of data block and in the Hadoop context, are perhaps most suited for cold-data storage.
Thanks!