Gvozden Nešković

Vectorized ZFS* RAIDZ Implementation

LUG 2016, Portland

neskovic@compeng.uni-frankfurt.de

* All trademarks are the property of their respective owners.
FAIR/GSI

• Facility for Antiproton and Ion Research
  - Linear and Ring particle accelerators
  - Heavy Ion Experiments
  - Medical irradiation facility for cancer therapy
  - Participation in the ALICE experiment at CERN

• HPC at FAIR/GSI
  - Green IT Cube data center
  - Compute clusters:
    - Prometheus (~9000 cores, QDR IB)
    - Kronos (~8000 cores, FDR IB)
  - Lustre storage clusters:
    - Hera (~7PB, Lustre 1.8)
    - Nyx (~7PB, 45 OSSs, Lustre 2.5 on ZFS)
    - ~1TB/s per experiment (10MHz event rate for CBM)
Lustre* & ZFS Motivation

• Lustre on ldiskfs:
  ▪ Version of ext3/4
  ▪ Random writes limited by disk IOPS
  ▪ Availability: long offline fsck
  ▪ Reliability: hardware RAID controllers

• Lustre on ZFS:
  ▪ ZFS Transactional Object Layer
  ▪ Random writes limited by disk bandwidth (COW)
  ▪ Data & metadata checksums, compression, snapshots
  ▪ Availability: online scrubbing/resilvering
  ▪ Reliability: Replication, RAIDZ1/2/3
ZFS RAIDZ levels

• ZFS volume management:
  ▪ Striping
  ▪ Mirroring
  ▪ RAIDZ levels

• RAIDZ-1/2/3 levels:
  ▪ Error Correction Erasure scheme
  ▪ Specialized Reed-Solomon Codes
  ▪ Advanced Block layout
RAIDZ Parity

• Properties:
  - Based on Galois field $GF[2^8]$ generated with $p(x)=x^8+x^4+x^3+x^2+1$
  - Erasure code

\[ P = D_0 \oplus D_1 \oplus \cdots \oplus D_n \]
\[ Q = 2^0 \cdot D_0 \oplus 2^1 \cdot D_1 \oplus \cdots \oplus 2^n \cdot D_n \]
\[ R = 4^0 \cdot D_0 \oplus 4^1 \cdot D_1 \oplus \cdots \oplus 4^n \cdot D_n \quad \text{where} \quad 2 \equiv X^1 \]
\[ 4 \equiv X^2 \]

• Addition:
  - XOR operation
  - Efficient in scalar and vector

• Multiplication:
  - By 2 and 4
  - By a constant:
    - Using $\log$ and $\exp$ look-up tables
    - $c \cdot a = \exp\{ \log(c) + \log(a) \}$
RAIDZ Parity Generation

• RAIDZ-1 parity generation:
  ▪ Simple XOR parity (P code only)

• RAIDZ-2/3 parity generation:
  ▪ Transform the Q and R equations:

\[
Q = D_0 \oplus 2 \cdot (D_1 \oplus \ldots \oplus 2 \cdot (D_{n-1} \oplus 2 \cdot D_n))
\]

\[
R = D_0 \oplus 4 \cdot (D_1 \oplus \ldots \oplus 4 \cdot (D_{n-1} \oplus 4 \cdot D_n))
\]

  ▪ RAIDZ-2 requires fast GF multiplication by 2 (PQ codes)
  ▪ RAIDZ-3 requires fast GF multiplication by 2 and 4 (PQR codes)
RAIDZ Data Reconstruction

• Trivial when using only P parity

• Direct solving:
  ▪ Solve parity equations (matrix inversion method)
  ▪ Used in the original RAIDZ3 reconstruction
  ▪ Requires $n$ GF multiplications per word

\[
D_x = x_p \cdot P \oplus x_q \cdot Q \oplus x_0 \cdot D_0 \oplus \cdots \oplus x_{n-2} \cdot D_{n-2}
\]
\[
D_y = y_p \cdot P \oplus y_q \cdot Q \oplus y_0 \cdot D_0 \oplus \cdots \oplus y_{n-2} \cdot D_{n-2}
\]

• Solving using syndromes:
  ▪ Used in the original RAIDZ2 reconstruction
  ▪ Syndromes calculated first
    ○ Parity calculation with zeroed missing data
  ▪ Requires 1 to 5 GF multiplications per word

\[
P = P_{xy} \oplus D_x \oplus D_y
\]
\[
Q = Q_{xy} \oplus 2^x \cdot D_x \oplus 2^y \cdot D_y
\]
\[
D_x = a \cdot (P \oplus P_{xy}) \oplus b \cdot (Q \oplus Q_{xy})
\]
\[
D_y = D_x \oplus (P \oplus P_{xy})
\]
Vectorizing GF multiplication

- GF multiplication:
  \[ a \cdot b = (a \times b) \mod p \]
  \[ a \cdot b = L(a \times b) \oplus M(a, b) \]

  - Sum of carry-less multiplication and modulo parts \(^1\)
  - Computed efficiently using two lookup-tables \(^2\)
    - SSE variant computes 16 multiplication in parallel
    - AVX2 variant computes 32 multiplication in parallel

- Contributed implementations:
  - Scalar 32 and 64 bit
  - SSE 128bit
  - AVX2 256bit

<table>
<thead>
<tr>
<th>GF operation</th>
<th>Scalar</th>
<th>SSE</th>
<th>AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Addition</td>
<td>8</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Multiplication by 2/4</td>
<td>8/4</td>
<td>16</td>
<td>32</td>
</tr>
<tr>
<td>Multiplication</td>
<td>1</td>
<td>16</td>
<td>32</td>
</tr>
</tbody>
</table>

1) “Intel” Carry-Less Multiplication Instruction and its Usage for Computing the GCM Mode”
2) “Optimizing Galois Field arithmetic for diverse processor architectures and Applications”. K.Greenan et al. 2008
ZFS RAIDZ-2 Results

- RAIDZ-2
- 8 data disks
- 2 parity disks
- Generate PQ
- Reconstruct 2 disks
ZFS RAIDZ-3 Results

- RAIDZ-3
- 8 data disks
- 3 parity disks
- Generate PQR
- Reconstruct 3 disks
Profilng with **perf** and FlameGraph\(^1\)

## New RAIDZ Implementations speed-up

<table>
<thead>
<tr>
<th>RAIDZ operation</th>
<th>Scalar</th>
<th>SSE</th>
<th>AVX2</th>
</tr>
</thead>
<tbody>
<tr>
<td>P generate</td>
<td>2.2</td>
<td>2.4</td>
<td>2.6</td>
</tr>
<tr>
<td>P reconstruct</td>
<td>1.4</td>
<td>2.0</td>
<td>2.2</td>
</tr>
<tr>
<td>PQ generate</td>
<td>1.5</td>
<td>4.1</td>
<td>4.3</td>
</tr>
<tr>
<td>Q reconstruct</td>
<td>1.5</td>
<td>7.2</td>
<td>8.8</td>
</tr>
<tr>
<td>PQ reconstruct</td>
<td>1.2</td>
<td>4.7</td>
<td>7.1</td>
</tr>
<tr>
<td>PQR generate</td>
<td>1.4</td>
<td>5.6</td>
<td>8.8</td>
</tr>
<tr>
<td>R reconstruct</td>
<td>4.8</td>
<td>20.7</td>
<td>32.3</td>
</tr>
<tr>
<td>PR reconstruct</td>
<td>8.5</td>
<td>43.0</td>
<td>69.1</td>
</tr>
<tr>
<td>QR reconstruct</td>
<td>5.0</td>
<td>35.5</td>
<td>60.2</td>
</tr>
<tr>
<td>PQR reconstruct</td>
<td>5.9</td>
<td>50.1</td>
<td>85.8</td>
</tr>
</tbody>
</table>

**Speed-up** relative to the original RAIDZ methods

*All trademarks are the property of their respective owners.*
Summary & Future work

• Benefits of vectorized RAIDZ methods:
  • Faster parity generation
  • Faster recalculation of missing data
  • Shorter scrub and resilver times
  • Increased reliability
  • Decreased system acquiring and running costs

• Future work:
  • Test and verify the implementation\textsuperscript{1)}
  • Upstream to ZFS on Linux
  • Linear scrub…

\textsuperscript{1)} "A program can be made arbitrarily fast if you relax the requirement of correctness." - D.Knuth
The End

Thank you!

Questions?