StRAID: Stripe-threaded Architecture for Parity-based RAIDs with Ultra-fast SSDs

Shucheng Wang¹, Qiang Cao¹, Ziyi Lu¹, **Hong Jiang**², Jie Yao¹ and Yuanyuan Dong³

¹ Huazhong University of Science and Technology
² University of Texas Arlington
³ Alibaba Group
RAID Systems

• RAID (Redundant Array of Independent Disks) is widely used
  • Non-parity RAID:
    ➢ RAID-0 (striping) and RAID-1 (mirroring)
  • Parity-based RAID:
    ➢ RAID-4/5/6
    ➢ Balancing performance and reliability
    ➢ Read-modify-write nature
RAID Systems

• RAID (Redundant Array of Independent Disks) is widely used
  • Non-parity RAID:
    ➢ RAID-0 (striping) and RAID-1 (mirroring)
  • Parity-based RAID:
    ➢ RAID-4/5/6
    ➢ Balancing performance and reliability
    ➢ Read-modify-write nature

• Linux MD: popular software RAID component
  • Linux kernel module
  • No need for extra hardware
  • Compatible with various storages
SSD Storage Trend

- Modern SSD hardware delivers higher write throughput
Linux MD upon SSDs

• Motivational Test
  • RAIDs setup
    ✗ Non-parity: RAID-0 level
    ✗ **Parity-based:** RAID-5 (5+1) and RAID-6 (4+2) level
      - Enable the multi-worker mechanism

• SSD products

<table>
<thead>
<tr>
<th>Device Types</th>
<th>Products</th>
<th>Capacity</th>
<th>Stable Write Throughput (MB/s)</th>
<th>Stable Read Throughput (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SATA SSD</td>
<td>Samsung 860 PRO</td>
<td>512GB</td>
<td>500</td>
<td>510</td>
</tr>
<tr>
<td>NVMe SSD</td>
<td>Samsung 970 PRO</td>
<td>512GB</td>
<td>2200</td>
<td>3200</td>
</tr>
<tr>
<td>NVMe SSD</td>
<td>Samsung 980 PRO</td>
<td>1TB</td>
<td><strong>2600</strong></td>
<td>6900</td>
</tr>
</tbody>
</table>

> **14 GB/s total write bandwidth on six SSDs**
Multi-thread Write Scalability

- Parity-based RAIDs fail to scale for high-performance SSDs
  - Larger performance gap on fast SSDs
  - Full-stripe writes (1MB, without read-modify-write) still suffers

![Graphs showing throughput for SATA, NVMe (64KB I/O), and NVMe (1MB I/O) SSDs with RAID-0, RAID-5, and RAID-6 configurations.](image-url)
Multi-thread Write Scalability

- Parity-based RAIDs fail to scale for high-performance SSDs
  - A diminishing return in performance of the multi-worker mechanism
    - Throughput gains peak at +16 worker threads (WTs)
    - 5% decline with more WTs

![Graph showing performance contribution of the multi-worker mechanism](image-url)
Analysis of MD

• "N-for-all" processing model
  • Incoming block I/Os are temporarily stored in the Stripe Cache
    • Aggregate bios at the granularity of stripes
    • Use stripe_heads (SH) to maintain stripe states
    • Store SHs in stripe_lists
Analysis of MD

• "N-for-all" processing model
  • Incoming block I/Os are temporarily stored in the Stripe Cache
  • A set number of WTs asynchronously and non-exclusively handle stripe-write tasks
Analysis of MD

• MD’s concurrency control
  ➢ The device_lock in MD
    • A **spin-lock** shared between WTs
    • For updating shared structures
      (stripe_lists and metadata, etc.)
Analysis of MD

• MD’s concurrency control
  ➢ MD device lock
    • A spin-lock shared between WTs
    • For updating shared structures
      (stripe_lists and metadata, etc)
  ➢ Stripe-write workflow:
    • Multi-stage stripe processing
Analysis of MD

• MD’s concurrency control
  ➢ MD device lock
    • A spin-lock shared between WTs
    • For updating shared structures (stripe_lists and metadata, etc)
  ➢ Stripe-write workflow:
    • Multi-stage stripe processing
    • Four handling steps in each stage
      1. Fetch a SH from handle_list
Analysis of MD

• MD’s concurrency control
  ➢ MD device lock
    • A spin-lock shared between WTs
    • For updating shared structures (stripe_lists and metadata, etc)
  ➢ Stripe-write workflow:
    • Multi-stage stripe processing
    • Four handling steps in each stage
      1. Fetch a SH from handle_list
      2. Analyze stripe & device states
         - Use semaphores
         - Need rcu_read_lock

 Stripe states

<table>
<thead>
<tr>
<th></th>
<th>STRIPE_ACTIVE</th>
<th>STRIPE_INSYNC</th>
<th>R5_Wantread</th>
<th>R5_Wantwrite</th>
</tr>
</thead>
<tbody>
<tr>
<td>conf-&gt;device_lock</td>
<td>✓</td>
<td>×</td>
<td>✓</td>
<td>×</td>
</tr>
<tr>
<td>handle_list</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stripe_lists</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inactive_list</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...... (51 more)
Analysis of MD

• MD’s concurrency control
  ➢ MD device lock
    • A spin-lock shared between WTs
    • For updating shared structures (stripe_lists and metadata, etc)
  ➢ Stripe-write workflow:
    • Multi-stage stripe processing
    • Four handling steps in each stage
      1. Fetch a SH from handle_list
      2. Analyze stripe & device states
      3. Operations for handling stripe
      4. Release and insert the SH into a stripe_list

![Diagram showing MD concurrency control and stripe-write workflow]
Analysis of MD

• Breakdown of CPU cycles on critical functions and locks in WTs
  - CPU becomes the bottleneck on concurrency control
  - Few CPU cycles are used to drive I/Os → storage devices are underutilized
StRAID overview

• "One-for-one" processing model
  • Goals:
    ➢ Efficient CPU utilization
    ➢ Reduce partial-stripe-write penalty
  • Stripe-threaded architecture
    ➢ Dedicated WT for each stripe-write
      • Eliminate global lock
      • Reduce stripe state checking

Diagram:

- UD
- WT
- SSD Group

Disk Group:
- SSD 1
- SSD 2
- SSD 3
- SSD 4

StRAID
StRAID overview

• "One-for-one" processing model
• Goals:
  ➢ Efficient CPU utilization
  ➢ Address partial-stripe-write penalty
• Stripe-threaded architecture
  ➢ Dedicated WT for each stripe-write
    • Eliminate global lock contention
    • Reduce stripe state checking
  ➢ Stripe State Table
    • Conduct thread collaboration
    • Maintain indispensable shared stripe states and per-stripe locks
**StRAID overview**

- "One-for-one" processing model
  - Goals:
    - Efficient CPU utilization
    - Address partial-stripe-write penalty
  - Stripe-threaded architecture
    - Dedicated WT for each stripe-write
      - Eliminate global lock contention
      - Reduce stripe state checking
    - Stripe State Table
      - Conduct thread collaboration
      - Maintain indispensable shared stripe states and per-stripe locks
    - Two-phase stripe submission
      - Opportunistic write batching
      - Per-stripe batching queue
StRAID's Concurrency Control

- **An example:** four I/O threads issue block I/Os
  - \(\text{bio 0} \to \text{stripe 5}\)
  - \(\text{bio 1 to bio 3} \to \text{stripe 8}\)

<table>
<thead>
<tr>
<th>Stripe ID</th>
<th>Stripe Lock</th>
<th>TID</th>
<th>is_batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Locked</td>
<td>0</td>
<td>True</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>Locked</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the batching phase.
StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the **batching phase**

### Stripe State Table

<table>
<thead>
<tr>
<th>Stripe ID</th>
<th>Stripe Lock</th>
<th>TID</th>
<th>is_batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Locked</td>
<td>0</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>8</td>
<td>Locked</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**WT0**

- bio 0 arriving
- Lock S5 (True)
- Read & Batch

**WT1**

- bio 1 arriving
- Lock S8 (True)
- Read & Batch

**WT2**

- bio 2 arriving
- Lock S8 (False)
- S8 Batch (True)

**WT3**

- **S5's batching phase**

- **S8's batching phase**
StRAID's Concurrency Control

- Dedicated WT aggregates requests targeting the same stripe in the batching phase

**Stripe State Table**

<table>
<thead>
<tr>
<th>Stripe ID</th>
<th>Stripe Lock</th>
<th>TID</th>
<th>is_batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Locked</td>
<td>0</td>
<td>True</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>8</td>
<td>Locked</td>
<td>1</td>
<td>True</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
StRAID's Concurrency Control

• Dedicated WT stops batching phase after reading complete
StRAID's Concurrency Control

- Dedicated WT stops batching phase after reading complete
- Requests failed to batch must **wait** for the dedicated WT to complete
StRAID's Concurrency Control

- After completing stripe processing, WT cleans up SST-entry and returns I/O
StRAID's Concurrency Control

- The waiting WT will try to re-acquire the stripe lock

---

**Stripe State Table**

<table>
<thead>
<tr>
<th>Stripe ID</th>
<th>Stripe Lock</th>
<th>TID</th>
<th>is_batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Unlocked</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>8</td>
<td>Locked</td>
<td>3</td>
<td>True</td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation Setup

• Platform

<table>
<thead>
<tr>
<th>System</th>
<th>Linux kernel version 5.13</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Duel Intel Xeon Gold 6328 CPU (totally 56 physical cores)</td>
</tr>
<tr>
<td>Memory</td>
<td>256 GB</td>
</tr>
<tr>
<td>SSD</td>
<td>6 x Samsung 970PRO (NVMe, 2.2GB/s stable write)</td>
</tr>
<tr>
<td></td>
<td>6 x Samsung 980PRO (NVMe, 2.6GB/s stable write)</td>
</tr>
<tr>
<td>PM</td>
<td>6 x Intel Optane PM 128GB (2.3GB/s stable write)</td>
</tr>
</tbody>
</table>

• RAID setup

- RAID-5 (5+1) and RAID-6 (4+2) with 64KB chunk size

• Workloads

- Micro-benchmarks: partial-stripe writes (64KB) and full-stripe writes (1MB)
- Macro-benchmarks: traces from Microsoft, Ali-Pangu, and Filebench
Micro-benchmark Results

- StRAID archives 2.4x - 3.1x higher write throughput than MD
- StRAID reduces 76% - 90% average write latency from MD
Breakdown of CPU cycles

- StRAID reduces up to 90% lock overhead
  - < 5% CPU overhead on the two-phase submission
- The total CPU utilization of StRAID is up to 6.3x lower than MD
• Run StRAID over six ramdisks (*-RAM) and Intel Optane PMs (*-PM)
• StRAID on RAMs archives up to 5.8x higher throughput than MD
Macro-benchmark Results

• Average throughput: 2x – 2.8x higher than MD
• Mean, average, and 99\textsuperscript{th}-percentile latency: 10.3x, 49%, and 25% lower than MD
Conclusion

• StRAID: a new architecture for parity-based RAID on SSDs
  ➢ Stripe-threaded architecture to efficiently parallelize stripe-write tasks
  ➢ Two-phase stripe submission to address partial-stripe-write penalty
  ➢ Performs significantly better than existing Linux MD

• See paper for more details

• Source code: https://github.com/wsczq1/straid

Thanks