RAID Performance Optimization

Young Jin Nam

System Software Laboratory
Department of Computer Science and Engineering
Pohang University of Science and Technology, KOREA
http://sslab.postech.ac.kr
Part II: Storage Performance Optimization

- Introduction
- Adaptive Thresholding (destaging scheme)
- Lazy Parity Update (write I/O perf. of RAID6)
- Selective Hole Plugging (pseudo sequential read I/O)
Introduction

- **Storage Optimization Points**
  - Efficient destaging scheme for fast write
  - Better I/O performance for RAID6 writes
  - Better I/O performance for pseudo sequential reads
Part II: Storage Performance Optimization

- Introduction
- **Adaptive Thresholding**  (destage scheme)
- Lazy Parity Update  (write I/O perf. of RAID6)
- Selective Hole Plugging  (pseudo sequential read I/O)
Adaptive Thresholding

- **Fast Writes using Non-Volatile Cache**
  - Write data to NV cache (fast write)
  - Flush it to disk (destaging)

- **Design Considerations for Destaging Process**
  - Determine *when* the destage process gets started/stopped
  - Choose *which* dirty blocks would be destaged into disks
Adaptive Thresholding

Existing Destage Algorithms

- High/Low Water Mark (HLWM) Algorithm [Menon’93]
  - start destaging if current cache occup ≥ HWM
  - stop destaging if current cache occup ≤ LWM
  - simple; widely used

- Linear Threshold (LT) Algorithm [Varma’95/’98]
  - multiple time-invariant tune-on/off thresholds
  - faster destaging rate with a higher threshold
  - high run-time overhead
Adaptive Thresholding

- **Drawback of HLWM algorithm**
  - Two threshold values are configured with the best knowledge of its initial information of I/O workload pattern & storage performance.
  - Initial I/O workloads are subject to change due to numerous reasons, such as increased storage usage, F/S fragmentations, etc.
  - But, the threshold values remain unchanged (time-invariant).
    - \( HWM: H_{static} = \rho_H S \)
    - \( LWM: L_{static} = \rho_L S \)

⇒ We need to devise a scheme to **automatically reconfigure the threshold values** according to any change with respect to the initial configuration.
Adaptive Thresholding

- Key Idea (1 of 2)
  - Dynamically adjust the two thresholds according to the changed workload patterns:
    - HWM: \( H_{\text{adaptive}}(t) = \rho_H(t)S \)
    - LWM: \( L_{\text{adaptive}}(t) = \rho_L(t)S \)

\[
T_H = \left(1 - \frac{\rho_H}{i_B}\right)S, \quad T_L = \frac{\rho_L d_B}{d_B}
\]

\[
\rho_H(t) = \max\{0, \min\{1, 1 - (1 - \rho_H) \frac{i(t)}{i_B}\}\}
\]

\[
\rho_L(t) = \min\{1, \rho_H, \max\{0, \rho_L \frac{d(t)}{d_B}\}\}
\]

\(0 \leq \rho_L(t) \leq \rho_H(t) \leq 1\)

If \(i(t) = i_B\) and \(d(t) = d_B\), then the proposed algorithm degenerates to the HLWM algorithm
Key Idea (2 of 2)

- Dynamically adjust the two thresholds according to the changed workload patterns

Case 1 & 3: Increased overwrite ratio
Reduced disk I/O traffic

Case 2 & 4: More burst-tolerable
Performance Evaluations

- Synthetic I/O workloads (1/2)
  - initially, 50% & 10%
  - dynamically adjusting HWM/LWM according to the changed workload patterns

<table>
<thead>
<tr>
<th>time period</th>
<th>avg. inter-arrival time</th>
<th>% of entire disk to be accessed</th>
<th>associated workload pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>20 msec</td>
<td>0.4%</td>
<td>initial</td>
</tr>
<tr>
<td>$T_2$</td>
<td>40 msec</td>
<td>0.4%</td>
<td>Case 1</td>
</tr>
<tr>
<td>$T_3$</td>
<td>20 msec</td>
<td>0.4%</td>
<td>initial</td>
</tr>
<tr>
<td>$T_4$</td>
<td>10 msec</td>
<td>0.4%</td>
<td>Case 2</td>
</tr>
<tr>
<td>$T_5$</td>
<td>20 msec</td>
<td>0.4%</td>
<td>initial</td>
</tr>
<tr>
<td>$T_6$</td>
<td>20 msec</td>
<td>0.2%</td>
<td>Case 3</td>
</tr>
<tr>
<td>$T_7$</td>
<td>20 msec</td>
<td>0.4%</td>
<td>initial</td>
</tr>
<tr>
<td>$T_8$</td>
<td>20 msec</td>
<td>0.5%</td>
<td>Case 4</td>
</tr>
</tbody>
</table>
Performance Evaluations

- Synthetic I/O workloads (2/2)
  - overwrite ratio: increased by 56% (Case1) & 36% (Case3)
  - disk I/O traffic: improved by 9% (Case1) & 4% (Case3)
  - lower overwrite ratio (Case2,4); little difference in disk I/O traffic
Performance Evaluations

- Traced I/O workloads (1/2)
  - initially, 50% & 10%
  - automatically obtain the reference values \((i_B, d_B)\)
  - dynamically adjust HWM/LWM according to the changed workload patterns

<table>
<thead>
<tr>
<th>type</th>
<th>traced period</th>
<th>read ratio</th>
<th>avg. req. size</th>
<th>avg. inter-arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>(W^1_{\text{traced}})</td>
<td>05/04/92 10am–12am</td>
<td>75.19%</td>
<td>6.5KB</td>
<td>21.10msec</td>
</tr>
<tr>
<td>(W^2_{\text{traced}})</td>
<td>06/15/92 10am–12am</td>
<td>17.86%</td>
<td>7.2KB</td>
<td>34.47msec</td>
</tr>
</tbody>
</table>
Performance Evaluations

- Traced I/O workloads (2/2)
  - with $W_{\text{trace}1}$, overwrite ratio is increased by 6.2%
  - with $W_{\text{trace}1}$, disk I/O traffic is reduced by 12%
  - with $W_{\text{trace}2}$, overwrite ratio & disk I/O traffic remain the same due to heavy I/O workload
Part II: Storage Performance Optimization

- Introduction
- Adaptive Thresholding (destaging scheme)
- Lazy Parity Update (write I/O perf. of RAID6)
- Selective Hole Plugging (pseudo sequential read I/O)
Why Write I/O is Slow in RAID6?

- Updating two parity data per each write

RAID 5 ⇒ 4 disk I/O accesses
(2 reads + 2 writes)

RAID 6 ⇒ 6 disk I/O accesses
(3 reads + 3 writes)
Lazy Parity Update

How to Enhance RAID6 Write I/O Performance?

- Exploiting numerous techniques developed for RAID5
  - parity logging, fast write using NVRAM, etc...
  - while assuring high reliability, this approach has some limit in its enhancement, we believe

- Sacrificing “too-high” reliability for better performance
  - deferring the update of the additional parity information
  - in case of RAID5, originally proposed by Savage & Wilkes – “A Frequently Reliable Array of Independent Disks (AFRAID)”

⇒ Too straightforward to apply it for a RAID6 architecture
Key Idea (1 of 2)
- Delaying the processing of updating a parity data
- Exploiting system idleness to process the delayed parity update
- Tolerating a single disk failure at any time (with FPG)
Lazy Parity Update

Key Idea (2 of 2)

- System state transition

![State Transition Diagram]

Expected Performance Results

<table>
<thead>
<tr>
<th></th>
<th>Write I/O performance</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Normal</td>
<td>1 disk failure</td>
</tr>
</tbody>
</table>

System Software Laboratory, POSTECH
Lazy Parity Update

- Underlying RAID6 Architecture – RM2

Data & Parity Layout of RM2
Lazy Parity Update

- Determination of FPG & BPG in a Single Stripe Group
  - Algorithm description

```
input : N disks, M stripe group size
output : FPG, BPG

begin
  /* each stripe group k has a set of parity stripe units */
  /* a stripe group denoted by PG={PG_i|0 \leq i \leq N-1} */

  if N is even then
    FPG = \{PG_{2i} | 0 \leq i < \frac{N}{2}\};
  else
    FPG = \{PG_{2i} | 0 \leq i < \frac{N}{2}\} \cup \{PG_{2j+1} | 0 \leq j < \frac{M-2}{2}\} \cup \{PG_{N-2k} | 0 \leq k < \frac{M-2}{2}\};
  endif

  BPG = PG - FPG;
end
```
Lazy Parity Update

- Determination of FPG & BPG in Multiple Stripe Groups
  - Rotating data/parity layout to evenly distribute I/O workload over multiple disks

```
          Stripe Group_0
          D_{2,3} D_{3,4} D_{4,5} D_{5,6} D_{6,7} D_{7,0} D_{0,1} D_{1,2}
          D_{1,4} D_{2,5} D_{3,6} D_{4,7} D_{5,0} D_{6,1} D_{7,2} D_{0,3}
          P_7   P_0   P_1   P_2   P_3   P_4   P_5   P_6   P_7

          Stripe Group_1
          D_{1,2} D_{2,3} D_{3,4} D_{4,5} D_{5,6} D_{6,7} D_{7,0} D_{0,1}
          D_{0,3} D_{1,4} D_{2,5} D_{3,6} D_{4,7} D_{5,0} D_{6,1} D_{7,2}

          . . . .

          Stripe Group_7
          D_{3,4} D_{4,5} D_{5,6} D_{6,7} D_{7,0} D_{0,1} D_{1,2} D_{2,3}
          D_{2,5} D_{3,6} D_{4,7} D_{5,0} D_{6,1} D_{7,2} D_{0,3} D_{1,4}
```
## I/O Performance Analysis (1 of 3)

- **Case 1: Small write (read-modify-write)**

\[
D_{sw}^\text{prop} = \begin{cases} 
4 & \text{if } N = \text{even} \\
4 + \frac{2(2M - 3)}{N} & \text{if } N = \text{odd}
\end{cases}
\]

\[
D_{\text{RAID6}}^\text{sw} = 6
\]

\[
\text{perf\_gain} = \begin{cases} 
1.5 & \text{if } N = \text{even} \\
\frac{3N}{2N + 2M - 3} & \text{if } N = \text{odd}
\end{cases}
\]

- **Case 2: Large write (reconstruction write)**

\[
D_{lw}^\text{prop} = \begin{cases} 
2M - 1 & \text{if } N = \text{even} \\
2M - 1 + \frac{6(M - 1)(2M - 3)}{N} & \text{if } N = \text{odd}
\end{cases}
\]

\[
D_{\text{RAID6}}^\text{lw} = 8(M - 1) + 1
\]

\[
\text{perf\_gain} = \text{approx. } 4 \text{ times if } N = \text{even}
\]
Lazy Parity Update

I/O Performance Analysis (2 of 3)

- Write I/O performance with synthetic random I/O workload
  - small/medium writes (8~32KB, 64KB): 2.0~1.2 times of RAID6
  - large writes (>128KB): 2.6~1.3 times of RAID6

---

# of disks (N) = 8

![Graph showing I/O performance with 8 disks]

# of disks (N) = 7

![Graph showing I/O performance with 7 disks]
I/O Performance Analysis (3 of 3)

- Write I/O performance with traced I/O workload (*cello* system)
  - $W_r$: RT of LPU is 1.45 times better than RAID6 (1.67 times at max.)
  - $W_w$: RT of LPU is 1.58 times better than RAID6 (2.13 times at max.)

<table>
<thead>
<tr>
<th>Type</th>
<th>Traced date/time</th>
<th># of I/Os</th>
<th>Write ratio</th>
<th>Average request size</th>
<th>Average inter-arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$W_r$</td>
<td>05/04/92, 10am–12pm</td>
<td>170569</td>
<td>24.81%</td>
<td>6.5KB</td>
<td>42.20msec</td>
</tr>
<tr>
<td>$W_w$</td>
<td>06/15/92, 10am–12pm</td>
<td>104453</td>
<td>82.14%</td>
<td>7.3KB</td>
<td>68.94msec</td>
</tr>
</tbody>
</table>
Reliability Analysis (1 of 3)

- MTTDL (Mean Time To Data Loss) analysis
  - assume that $1/\lambda_1 = 1,000,000$ hours, $1/\mu_1 = 48$ hours
  - $\lambda_2$, $\mu_2$, and $\mu_3$ are empirically obtained by regenerating traced I/O workloads

\[
\begin{align*}
\text{NORM}_{\text{safe}} & \xrightarrow{N\lambda_1} \text{DEG1}_{\text{safe}} \\
\text{DEG1}_{\text{safe}} & \xrightarrow{(N-1)\lambda_1} \text{DEG2} \\
\text{DEG2} & \xrightarrow{(N-2)\lambda_1} \text{FAIL} \\
\text{NORM}_{\text{unsafe}} & \xrightarrow{N\lambda_1} \text{DEG1}_{\text{unsafe}} \\
\text{DEG1}_{\text{unsafe}} & \xrightarrow{(N-1)\lambda_1} \\
\end{align*}
\]
Lazy Parity Update

- Reliability Analysis (2 of 3)
  - MTTDL (Mean Time To Data Loss) analysis
    - using the fundamental matrix $M = [I - Q]^{-1}$, where $I$ is an identical matrix and $Q$ is a truncated stochastic transitional probability matrix
    - $MTTDL = \sum_{j=1}^{n} \{m_{ij} \text{ of } M\}$

\[
Q = \begin{bmatrix}
1 - N\lambda_1 - \lambda_2 & N\lambda_1 & 0 & \lambda_2 & 0 \\
\mu_1 & 1 - (N-1)\lambda_1 - \mu_1 & (N-1)\lambda_1 & 0 & 0 \\
\mu_1 & 0 & 1 - \mu_1 - (N-2)\lambda_1 & 0 & 0 \\
\mu_2 & 0 & 0 & 1 - N\lambda_1 - \mu_2 & N\lambda_1 \\
0 & \mu_3 & 0 & 0 & 1 - \mu_3 - (N-1)\lambda_1
\end{bmatrix}
\]

\[
MTTDL_{\text{proposed}} = \frac{\alpha}{N(N-1)(N-2)\lambda_1^3} + \frac{\beta}{\gamma N(N-1)(N-2)\lambda_1^3},
\]

where $\alpha = (2\lambda_1^2 - 6N\lambda_1^2 + 3N^2\lambda_1^2 - 3\lambda_1\mu_1 + 3N\lambda_1\mu_1 + \mu_2^2), \beta = (2N\lambda_1\lambda_2 - 5N^2\lambda_1^2\lambda_2 + 4N^3\lambda_1^2\lambda_2 - N^4\lambda_1^4\lambda_2 + 6N^2\lambda_2\lambda_1\mu_1 - 18N\lambda_2\lambda_1\mu_1 + 17N^2\lambda_2\lambda_1\mu_1 - 5N^3\lambda_2^2\lambda_1\mu_1 - 11\lambda_2^2\lambda_2\mu_1^2 + 19N\lambda_2^2\lambda_2\mu_1^2 - 8N^2\lambda_2^2\lambda_2\mu_1^2 + 6\lambda_1\lambda_2\mu_1^2 - 5N\lambda_1\lambda_2\mu_1^2 - \lambda_2\mu_1^2), \gamma = (2N\lambda_1^3 - 3N^2\lambda_1^3 + N^3\lambda_1^3 + 2\lambda_1^2\lambda_2 - 3N\lambda_1^2\lambda_2 + N\lambda_1^2\lambda_2 - 3\lambda_1\lambda_2\mu_1 + 2N\lambda_1\lambda_2\mu_1 + \lambda_2\mu_1^2 + 2\lambda_1^2\mu_2 - 3N\lambda_1^2\mu_2 + N^2\lambda_2^2\mu_2 - 2N\lambda_1^2\mu_3 + N^2\lambda_2^2\mu_3 - 2\lambda_1\lambda_2\mu_3 + N\lambda_1\lambda_2\mu_3 - 2\lambda_1\mu_2\mu_3 + N\lambda_1\mu_2\mu_3)
Lazy Parity Update

Reliability Analysis (3 of 3)

- Results of MTTDL (Mean Time To Data Loss) analysis
  - 0.88~0.79 times of RAID6 (=1,725~1,553 times of RAID5)

<table>
<thead>
<tr>
<th>Workload</th>
<th>1/λa</th>
<th>1/μa</th>
<th>1/μb</th>
<th>α</th>
<th>Proposed</th>
<th>RM2</th>
<th>RAID5</th>
</tr>
</thead>
<tbody>
<tr>
<td>W_r(x1)</td>
<td>0.187</td>
<td>0.013</td>
<td>0.047</td>
<td>0.00056</td>
<td>6.42E+11</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_r(x2)</td>
<td>0.117</td>
<td>0.030</td>
<td>0.206</td>
<td>0.00242</td>
<td>2.67E+11</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_r(x4)</td>
<td>0.052</td>
<td>0.089</td>
<td>0.872</td>
<td>0.01023</td>
<td>3.66E+10</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_r(x8)</td>
<td>0.022</td>
<td>0.092</td>
<td>2.577</td>
<td>0.03021</td>
<td>8.49E+09</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_w(x1)</td>
<td>0.047</td>
<td>0.005</td>
<td>0.063</td>
<td>0.00074</td>
<td>5.78E+11</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_w(x2)</td>
<td>0.026</td>
<td>0.006</td>
<td>0.263</td>
<td>0.00309</td>
<td>2.28E+11</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_w(x4)</td>
<td>0.076</td>
<td>0.064</td>
<td>1.467</td>
<td>0.01720</td>
<td>2.57E+10</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
<tr>
<td>W_w(x8)</td>
<td>0.080</td>
<td>0.080</td>
<td>5.104</td>
<td>0.05982</td>
<td>4.78E+09</td>
<td>7.28E+11</td>
<td>3.72E+8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Traced date/time</th>
<th># of I/Os</th>
<th>Write ratio</th>
<th>Average request size</th>
<th>Average inter-arrival time</th>
</tr>
</thead>
<tbody>
<tr>
<td>W_r</td>
<td>05/04/92, 10am−12pm</td>
<td>170569</td>
<td>24.81%</td>
<td>6.5KB</td>
<td>42.20msec</td>
</tr>
<tr>
<td>W_w</td>
<td>06/15/92, 10am−12pm</td>
<td>104453</td>
<td>82.14%</td>
<td>7.3KB</td>
<td>68.94msec</td>
</tr>
</tbody>
</table>
Part II: Storage Performance Optimization

- Introduction
- Adaptive Thresholding  (destage scheme)
- Lazy Parity Update  (write I/O perf. of RAID6)
- Selective Hole Plugging  (pseudo sequential read I/O)
Selective Hole Plugging

- **Strict Sequential Read**
  - No jumps or holes
  - Prefetch is "always" activated (with two I/Os in ST373307FC disk)

- **Pseudo Sequential Read**
  - Jumps (or holes) in sequential runs
  - Prefetch is "selectively" turned on depending on req_size & jmp_cnt
  - Represented by
    \{S(size), R(run), J(jmp_cnt), think_time\}
Selective Hole Plugging

- Abstracted Model for a Pseudo Sequential Read

![Diagram]

μ is time-variant
- 1. head switch time (trk/cyl)
- 2. zoned bit recording
- 3. prefetch policy unknown
Selective Hole Plugging

What if $\lambda > \mu$ with Strict Sequential Read?

- At the worst case, read hit on the on-going prefetch $\Rightarrow$ no extra rotational delay to locate its start_block_address (only req_sz/BWcache)
- Prefetch is "always" turned on

What if $\lambda > \mu$ with Pseudo Sequential Read?

- Read miss may occur while prefetching unwanted data
- Abort the current prefetch activity $\Rightarrow$ experiencing a long rotational latency to locate its start_block_address
- Deactivate prefetch $\Rightarrow$ not exploiting prefetch until it's re-activated with a next segment

For a given $\lambda > \mu$, more performance degradation is expected with a pseudo sequential read
Selective Hole Plugging

- Problem Observation with ST373307FC Disk
  - Strict sequential read \((s=16\text{KB}, r=1, j=1, T_{\text{think}}=0)_{sr}\)
    \(\Rightarrow\) no spikes in response times
  - Pseudo sequential read \((s=16\text{KB}, r=1, j=2, T_{\text{think}}=0)_{sr}\)
    \(\Rightarrow\) spikes in response times every 12 requests
Selective Hole Plugging

Problem Description

- How to determine amount/location of delays?
- How to decide if it really improves the performance?

pseudo sequential read 
\((s, r_j > 1, T_{\text{think}})\)

prefetch-always-on
pseudo sequential read

disk

prefetch deactivation overhead (p)

injecting delays

3. checking delay overhead

determining delays

1. delay location

2. delay amount

passed

prefetch deactivation may occur!
Selective Hole Plugging

Key Idea

Transform a pseudo sequential read to its corresponding strict sequential read by replacing holes with extra read I/Os if the expected performance gain outweighs the overhead of extra read I/Os.

Transform a pseudo sequential read to its corresponding strict sequential read by replacing holes with extra read I/Os if the expected performance gain outweighs the overhead of extra read I/Os.
Selective Hole Plugging

- **Experimental Environment**
  - Hardware specifications
    - Intel Pentium4 1.5Ghz, 512MB SDRAM
    - QLogic QLA2200 FC HBA
    - Seagate ST373307FC disk
  - Synthetic I/O workload generator
    - using SCSI generic I/F
    - \( T_{\text{think}} = 0 \text{msec} \)

- **Performance Evaluations**
  - Observing the existence of high response time with \( s=16\text{KB},r=1,j=2,T_{\text{think}}=0 \)
  - No spikes with the proposed scheme

![Graph showing response time comparison]
Selective Hole Plugging

Performance Evaluations

- Obtained performance gain (1 of 2)
  - different request sizes under a fixed run count
    - with run_count=1 (benefit zone ⇒ 2 ≤ j ≤ 4)
      - 1.67~1.73 times performance gain at 32 KB & 64KB
    - with run_count=4 (benefit zone ⇒ 2 ≤ j ≤ 6)
      - 2.0 times performance gain at 16KB

Run count=1

Run count=4
Selective Hole Plugging

Performance Evaluations

- Obtained performance gain (2 of 2)
  - different run counts under a fixed request size
  - with req_size=16KB (benefit zone \(2 \leq j \leq 8\))
    - 2.0 times performance gain at \(j=2 \& r=4\)
  - with req_size=32KB (benefit zone \(2 \leq j \leq 5\))
    - 2.0 times performance gain at \(j=2 \& r=16\)
Part II – Summary

Our Contributions

- Proposed a technique called **Adaptive Thresholding**
  - to automatically reconfigure the two thresholds of HLWM destage algorithm according to the changed workloads

- Proposed a technique called **Lazy Parity Update**
  - to enhance the write I/O performance of a well-known RAID6 architecture, RM2 by delaying the processing of one parity until the storage system becomes idle

- Proposed a technique called **Selective Hole Plugging**
  - to improve the poor performance of a pseudo sequential read caused by the mismatch between prefetch operation of a disk and the read speed from the host system
Part II – Summary

Publications (1 of 2)


Part II – Summary

Publications (2 of 2)


Future Work

Part II: Storage Performance Optimization

- Predicting I/O workload for a storage server
  - at least, idleness (*not self-similar*) & burstness
  - better destage scheme for delayed write I/Os (idleness, burstness)
  - better scheme to process the delayed parity groups (idleness)
  - better power management scheme (idleness)

- Finding the “optimal” thresholds for HLWM destage algorithm
  - currently, these are configured by a system admin. with the best knowledge of the initial I/O workload patterns & storage performance
  - need to devise a scheme that automatically determines these values

- Expanding Selective Hole Plugging
  - instead of plugging all holes with extra reads, plugging part of all the holes with extra reads (better performance)
  - applying it into a disk array environment (PosRAID)
The End of Presentation