Adaptive Replacement Cache

N. Megiddo and D. S. Modha
IBM Almaden Research Center
"Computer Science has only three ideas: cache,

"

Greg Ganger, CMU
"Computer Science has only three ideas: cache, hash."

Greg Ganger, CMU
"Computer Science has only three ideas: cache, hash, trash."

Greg Ganger, CMU
"...constructing a hierarchy of memories, each of which has greater capacity ... but which is less quickly accessible."
von Neumann et al., 1946
"...constructing a hierarchy of memories, each of which has greater capacity ... but which is less quickly accessible."  
von Neumann et al., 1946

"Yea, from the table of my memory, I'll wipe away all trivial fond records"
Shakespeare in Hamlet, 1603
The Replacement Cache Problem

cache: fast but expensive        disks: cheap but slow

How to manage the cache?
Which page to replace?
How to achieve a high hit ratio?
A Brief Survey

- LRU (dates back to 1965, at least)
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant
A Brief Survey

- LRU (dates back to 1965, at least)
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant

- LFU (dates back to 1971, at least)
  - exploits "frequency"
  - is scan-resistant
  - logarithmic time complexity (per request)
  - periodic resizing required to prevent stale pages
  - does not capture "clustered locality of reference"
A Brief Survey

- **LRU (dates back to 1965, at least)**
  - constant time and space complexity & simple-to-implement
  - captures "clustered locality of reference"
  - does not exploit "frequency"
  - is not scan-resistant

- **LFU (dates back to 1971, at least)**
  - exploits "frequency"
  - is scan-resistant
  - logarithmic time complexity (per request)
  - periodic resizing required to prevent stale pages
  - does not capture "clustered locality of reference"

- **LRU + LFU:**
  - Log-complexity: LRU-2, LRFU
  - Constant-complexity in expected sense: LIRS, FBR
  - Unbounded Space complexity: LIRS
  - Difficulty of tuning: FBR, LRU-2, 2Q, SLRU, LRFU, LIRS
  - Stringent Assumption on Workload: MQ
Difficulty of Tuning

- Shasha and Johnson (1994):
  - "difficult to model the tunables of LRU-2"
- Lee et al. (1998):
  - "... parameters in 2Q ... need to be carefully tuned"
- Lee et al. (2002):
  - "... looking for ways to tune the parameters of LRFU"
  - reported LRFU results with best offline choices
- Wong and Wilkes (2002):
  - For SLRU "... the optimal size ... varied greatly with the workload"

The best tunable parameter values depend upon the workload and the cache size.
Cache Directory (Registry)

- $L_1$: pages were seen once recently ("recency")
Cache Directory (Registry)

- $L_1$: pages were seen once recently ("recency")
- $L_2$: pages were seen at least twice recently ("frequency")
Cache Directory (Registry)

- L₁: pages were seen once recently ("recency")
- L₂: pages were seen at least twice recently ("frequency")

If L₁ contains exactly c pages
  --replace the LRU page in L₁
else
  --replace the LRU page in L₂.
Cache Directory (Registry)

- L₁: pages were seen once recently ("recency")
- L₂: pages were seen at least twice recently ("frequency")
- If L₁ contains exactly c pages
  --replace the LRU page in L₁
  else
    --replace the LRU page in L₂.
- Lemma: The c most recent pages are in the union of L₁ and L₂.
Fixed Replacement Cache (FRC)

- Divide $L_1$ into $T_1$ (top) & $B_1$ (bottom)
Fixed Replacement Cache (FRC)

- Divide $L_1$ into $T_1$ (top) & $B_1$ (bottom)
- Divide $L_2$ into $T_2$ (top) & $B_2$ (bottom)
Fixed Replacement Cache (FRC)

- Divide $L_1$ into $T_1$ (top) & $B_1$ (bottom)
- Divide $L_2$ into $T_2$ (top) & $B_2$ (bottom)
- $T_1$ and $T_2$ contain $c$ pages
  - in cache and in directory
Fixed Replacement Cache (FRC)

- Divide L₁ into T₁ (top) & B₁ (bottom)
- Divide L₂ into T₂ (top) & B₂ (bottom)
- T₁ and T₂ contain c pages
  - in cache and in directory
- B₁ and B₂ contain c pages
  - in directory, but not in cache
Fixed Replacement Cache (FRC)

- Divide L₁ into T₁ (top) & B₁ (bottom)
- Divide L₂ into T₂ (top) & B₂ (bottom)
- T₁ and T₂ contain c pages
  - in cache and in directory
- B₁ and B₂ contain c pages
  - in directory, but not in cache
- FRC(p):
  - Set target size of T₁ to p
Fixed Replacement Cache (FRC)

- Divide \( L_1 \) into \( T_1 \) (top) & \( B_1 \) (bottom)
- Divide \( L_2 \) into \( T_2 \) (top) & \( B_2 \) (bottom)
- \( T_1 \) and \( T_2 \) contain \( c \) pages
  - in cache and in directory
- \( B_1 \) and \( B_2 \) contain \( c \) pages
  - in directory, but not in cache
- FRC(\( p \)):
  - Set target size of \( T_1 \) to \( p \)
  - If \( T_1 \) contains more than \( p \) pages,
    --replace LRU page in \( T_1 \),
  else
    --replace LRU page in \( T_2 \).
Adaptive Replacement Cache (ARC)

- Adapt target size of $T_1$ to an observed workload
Adaptive Replacement Cache (ARC)

- Adapt target size of $T_1$ to an observed workload
- A self-tuning algorithm:
  - hit in $T_1$ or $T_2$: do nothing
Adaptive Replacement Cache (ARC)

- Adapt target size of $T_1$ to an observed workload
- A self-tuning algorithm:
  - hit in $T_1$ or $T_2$: do nothing
  - hit in $B_1$: increase target of $T_1$
Adaptive Replacement Cache (ARC)

- Adapt target size of $T_1$ to an observed workload
- A self-tuning algorithm:
  - hit in $T_1$ or $T_2$: do nothing
  - hit in $B_1$: increase target of $T_1$
  - hit in $B_2$: decrease target of $T_1$
Adaptive Replacement Cache (ARC)

- Adapt target size of $T_1$ to an observed workload
- A self-tuning algorithm:
  - hit in $T_1$ or $T_2$: do nothing
  - hit in $B_1$: increase target of $T_1$
  - hit in $B_2$: decrease target of $T_1$
- ARC is scan-resistant
"recency"

L₁

"frequency"

L₂

midpoint
ARC has Low Computational Overhead

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>ARC</th>
<th>2Q</th>
<th>LRU-2</th>
<th>LRFU</th>
<th>LRFU</th>
<th>LRFU</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>17</td>
<td>14</td>
<td>17</td>
<td>33</td>
<td>554</td>
<td>408</td>
<td>28</td>
</tr>
<tr>
<td>2048</td>
<td>12</td>
<td>14</td>
<td>17</td>
<td>27</td>
<td>599</td>
<td>451</td>
<td>28</td>
</tr>
<tr>
<td>4096</td>
<td>12</td>
<td>15</td>
<td>17</td>
<td>27</td>
<td>649</td>
<td>494</td>
<td>29</td>
</tr>
<tr>
<td>8192</td>
<td>12</td>
<td>16</td>
<td>18</td>
<td>28</td>
<td>694</td>
<td>537</td>
<td>29</td>
</tr>
<tr>
<td>16384</td>
<td>13</td>
<td>16</td>
<td>19</td>
<td>30</td>
<td>734</td>
<td>418</td>
<td>30</td>
</tr>
<tr>
<td>32768</td>
<td>14</td>
<td>17</td>
<td>18</td>
<td>31</td>
<td>716</td>
<td>420</td>
<td>31</td>
</tr>
<tr>
<td>65536</td>
<td>14</td>
<td>16</td>
<td>18</td>
<td>32</td>
<td>648</td>
<td>424</td>
<td>34</td>
</tr>
<tr>
<td>131072</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>32</td>
<td>533</td>
<td>432</td>
<td>39</td>
</tr>
<tr>
<td>262144</td>
<td>13</td>
<td>13</td>
<td>14</td>
<td>30</td>
<td>427</td>
<td>435</td>
<td>42</td>
</tr>
<tr>
<td>524288</td>
<td>12</td>
<td>13</td>
<td>13</td>
<td>27</td>
<td>263</td>
<td>443</td>
<td>45</td>
</tr>
</tbody>
</table>

Trace P9
ARC Compares well to LRU, 2Q, LRU-2, LRFU, LIRS

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>ARC</th>
<th>2Q</th>
<th>LRU-2</th>
<th>LRFU</th>
<th>LIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONLINE</td>
<td>ONLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
</tr>
<tr>
<td>1024</td>
<td>4.09</td>
<td>4.16</td>
<td>4.13</td>
<td>4.07</td>
<td>4.09</td>
<td>4.08</td>
</tr>
<tr>
<td>2048</td>
<td>4.84</td>
<td>4.89</td>
<td>4.89</td>
<td>4.83</td>
<td>4.84</td>
<td>4.83</td>
</tr>
<tr>
<td>4096</td>
<td>5.61</td>
<td>5.76</td>
<td>5.76</td>
<td>5.81</td>
<td>5.61</td>
<td>5.61</td>
</tr>
<tr>
<td>8192</td>
<td>6.22</td>
<td>7.14</td>
<td>7.52</td>
<td>7.54</td>
<td>7.29</td>
<td>6.61</td>
</tr>
<tr>
<td>16384</td>
<td>7.09</td>
<td>10.12</td>
<td>11.05</td>
<td>10.67</td>
<td>11.01</td>
<td>9.29</td>
</tr>
<tr>
<td>32768</td>
<td>8.93</td>
<td>15.94</td>
<td>16.89</td>
<td>16.36</td>
<td>16.35</td>
<td>15.15</td>
</tr>
<tr>
<td>65536</td>
<td>14.43</td>
<td>26.09</td>
<td>27.46</td>
<td>25.79</td>
<td>25.35</td>
<td>25.65</td>
</tr>
<tr>
<td>131072</td>
<td>29.21</td>
<td>38.68</td>
<td>41.09</td>
<td>39.58</td>
<td>39.78</td>
<td>40.37</td>
</tr>
<tr>
<td>262144</td>
<td>49.11</td>
<td>53.47</td>
<td>53.31</td>
<td>53.43</td>
<td>54.56</td>
<td>53.65</td>
</tr>
<tr>
<td>524288</td>
<td>60.91</td>
<td>63.56</td>
<td>61.64</td>
<td>63.15</td>
<td>63.13</td>
<td>63.89</td>
</tr>
</tbody>
</table>

Trace P12
ARC Compares well to LRU, MQ, and 2Q (all online)

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>MQ</th>
<th>2Q</th>
<th>ARC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONLINE</td>
<td>ONLINE</td>
<td>ONLINE</td>
<td>ONLINE</td>
</tr>
<tr>
<td>16384</td>
<td>0.20</td>
<td>0.20</td>
<td>0.73</td>
<td>1.04</td>
</tr>
<tr>
<td>32768</td>
<td>0.40</td>
<td>0.40</td>
<td>1.47</td>
<td>2.08</td>
</tr>
<tr>
<td>65536</td>
<td>0.79</td>
<td>0.79</td>
<td>2.85</td>
<td>4.07</td>
</tr>
<tr>
<td>131072</td>
<td>1.59</td>
<td>1.59</td>
<td>5.52</td>
<td>7.78</td>
</tr>
<tr>
<td>262144</td>
<td>3.23</td>
<td>4.04</td>
<td>10.36</td>
<td>14.30</td>
</tr>
<tr>
<td>524288</td>
<td>8.06</td>
<td>14.39</td>
<td>18.89</td>
<td>24.34</td>
</tr>
<tr>
<td>1048576</td>
<td>27.62</td>
<td>40.13</td>
<td>35.39</td>
<td>40.44</td>
</tr>
<tr>
<td>1572864</td>
<td>50.86</td>
<td>56.49</td>
<td>53.19</td>
<td>57.19</td>
</tr>
<tr>
<td>2097152</td>
<td>68.68</td>
<td>70.45</td>
<td>67.36</td>
<td>71.41</td>
</tr>
<tr>
<td>4194304</td>
<td>87.30</td>
<td>87.29</td>
<td>86.22</td>
<td>87.26</td>
</tr>
</tbody>
</table>

Trace MergeS
ARC outperforms LRU & is empirically universal

<table>
<thead>
<tr>
<th>Workload</th>
<th>Cache Size</th>
<th>LRU ONLINE</th>
<th>ARC ONLINE</th>
<th>FRC OFFLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>16</td>
<td>16.55</td>
<td>28.26</td>
<td>29.39</td>
</tr>
<tr>
<td>P2</td>
<td>16</td>
<td>18.47</td>
<td>27.38</td>
<td>27.61</td>
</tr>
<tr>
<td>P3</td>
<td>16</td>
<td>3.57</td>
<td>17.12</td>
<td>17.60</td>
</tr>
<tr>
<td>P4</td>
<td>16</td>
<td>5.24</td>
<td>11.24</td>
<td>9.11</td>
</tr>
<tr>
<td>P5</td>
<td>16</td>
<td>6.73</td>
<td>14.27</td>
<td>14.29</td>
</tr>
<tr>
<td>P6</td>
<td>16</td>
<td>4.24</td>
<td>23.84</td>
<td>22.62</td>
</tr>
<tr>
<td>P7</td>
<td>16</td>
<td>3.45</td>
<td>13.77</td>
<td>14.01</td>
</tr>
<tr>
<td>P8</td>
<td>16</td>
<td>17.18</td>
<td>27.51</td>
<td>28.92</td>
</tr>
<tr>
<td>P9</td>
<td>16</td>
<td>8.28</td>
<td>19.73</td>
<td>20.28</td>
</tr>
<tr>
<td>P10</td>
<td>16</td>
<td>2.48</td>
<td>9.46</td>
<td>9.63</td>
</tr>
<tr>
<td>P11</td>
<td>16</td>
<td>20.92</td>
<td>26.48</td>
<td>26.57</td>
</tr>
<tr>
<td>P12</td>
<td>16</td>
<td>8.93</td>
<td>15.94</td>
<td>15.97</td>
</tr>
<tr>
<td>P13</td>
<td>16</td>
<td>7.83</td>
<td>16.60</td>
<td>16.81</td>
</tr>
<tr>
<td>P14</td>
<td>16</td>
<td>15.73</td>
<td>20.52</td>
<td>20.55</td>
</tr>
<tr>
<td>Concat</td>
<td>16</td>
<td>14.38</td>
<td>21.67</td>
<td>21.63</td>
</tr>
<tr>
<td>Merge</td>
<td>128</td>
<td>38.05</td>
<td>39.91</td>
<td>39.40</td>
</tr>
<tr>
<td>DS1</td>
<td>1024</td>
<td>11.65</td>
<td>22.52</td>
<td>18.72</td>
</tr>
<tr>
<td>SPC1</td>
<td>4096</td>
<td>9.19</td>
<td>20.00</td>
<td>20.11</td>
</tr>
<tr>
<td>MergeS</td>
<td>4096</td>
<td>27.62</td>
<td>40.44</td>
<td>40.18</td>
</tr>
</tbody>
</table>
Summary: ARC

- Low compute overhead
Summary: ARC

- Low compute overhead
- Low space overhead
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice
- Significantly outperforms LRU on all workloads examined
Summary: ARC

- Low compute overhead
- Low space overhead
- No periodic resizing
- Captures both "recency" and "frequency"
  - self-tuning: autonomically adapts to evolving workloads and relentlessly balances between recency and frequency
  - empirically universal
- Scan-resistant
- Comparable to state-of-the-art algorithms with best offline choice
- Significantly outperforms LRU on all workloads examined
- Requires only a handful of lines of code! See:
  - "ARC: A Self-tuning, low overhead Replacement Cache"
  - "One Up on LRU"
## At-a-glance Comparision

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Compute Overhead</th>
<th>Space Overhead</th>
<th>Self-Tuning</th>
<th>Scan Resistant</th>
<th>No Re-sizing</th>
</tr>
</thead>
<tbody>
<tr>
<td>LRU</td>
<td>constant</td>
<td>1x</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>LFU</td>
<td>log</td>
<td>1x</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>LRU-2</td>
<td>log</td>
<td>1x-2x</td>
<td>No</td>
<td>Depends</td>
<td>Yes</td>
</tr>
<tr>
<td>2Q</td>
<td>constant</td>
<td>1x-2x</td>
<td>No</td>
<td>Depends</td>
<td>Yes</td>
</tr>
<tr>
<td>LIRS</td>
<td>constant (E)</td>
<td>unbounded</td>
<td>No</td>
<td>Depends</td>
<td>Yes</td>
</tr>
<tr>
<td>LRFU</td>
<td>log</td>
<td>1x-2x</td>
<td>No</td>
<td>Depends</td>
<td>Yes</td>
</tr>
<tr>
<td>FBR</td>
<td>constant (E)</td>
<td>1x</td>
<td>No</td>
<td>Depends</td>
<td>No</td>
</tr>
<tr>
<td>ARC</td>
<td>constant</td>
<td>2x</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## P8: LRU, 2Q, LRU-2, LRFU, LIRS

<table>
<thead>
<tr>
<th>Cache Size</th>
<th>LRU</th>
<th>ARC</th>
<th>2Q</th>
<th>LRU-2</th>
<th>LRFU</th>
<th>LIRS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ONLINE</td>
<td>ONLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
<td>OFFLINE</td>
</tr>
<tr>
<td>1024</td>
<td>0.35</td>
<td>1.22</td>
<td>0.94</td>
<td>1.63</td>
<td>0.69</td>
<td>0.79</td>
</tr>
<tr>
<td>2048</td>
<td>0.45</td>
<td>2.43</td>
<td>2.27</td>
<td>3.01</td>
<td>2.18</td>
<td>1.71</td>
</tr>
<tr>
<td>4096</td>
<td>0.73</td>
<td>5.28</td>
<td>5.13</td>
<td>5.50</td>
<td>3.53</td>
<td>3.60</td>
</tr>
<tr>
<td>8192</td>
<td>2.30</td>
<td>9.19</td>
<td>10.27</td>
<td>9.87</td>
<td>7.58</td>
<td>7.67</td>
</tr>
<tr>
<td>16384</td>
<td>7.37</td>
<td>16.48</td>
<td>18.78</td>
<td>17.18</td>
<td>14.83</td>
<td>15.26</td>
</tr>
<tr>
<td>32768</td>
<td>17.18</td>
<td>27.51</td>
<td>31.33</td>
<td>28.86</td>
<td>28.37</td>
<td>27.29</td>
</tr>
<tr>
<td>65536</td>
<td>36.10</td>
<td>43.42</td>
<td>47.61</td>
<td>45.77</td>
<td>46.72</td>
<td>45.36</td>
</tr>
<tr>
<td>131072</td>
<td>62.10</td>
<td>66.35</td>
<td>69.45</td>
<td>67.56</td>
<td>66.60</td>
<td>69.65</td>
</tr>
<tr>
<td>262144</td>
<td>89.26</td>
<td>89.28</td>
<td>88.92</td>
<td>89.59</td>
<td>90.32</td>
<td>89.78</td>
</tr>
<tr>
<td>524288</td>
<td>96.77</td>
<td>97.30</td>
<td>96.16</td>
<td>97.22</td>
<td>97.38</td>
<td>97.21</td>
</tr>
</tbody>
</table>
Cache Directory (Registry)

1: if (L1->hit(page)) {
2:     L1->delete(page);
3:     L2->insert_mru(page);
4: }
5: else if (L2->hit(page)) {
6:     L2->delete(page);
7:     L2->insert_mru(page);
8: }
9: else if (L1->length() == c) {
10:     L1->delete_lru();
11:     L1->insert_mru(page);
12: }
13: else {
14:     if (L1->length() + L2->length() == 2*c) {
15:         L2->delete_lru();
16:     }
17:     L1->insert_mru(page);
18: }
Applications

- Storage controllers
- File Systems
- Operating Systems
- Disks, RAID
- Databases
- Middleware
- Web Caching, Search Query Caching
- Micro-Processors
- Data Compression
## ARC outperforms LRU

<table>
<thead>
<tr>
<th>Workload</th>
<th>Cache Size</th>
<th>LRU ONLINE</th>
<th>ARC ONLINE</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>16</td>
<td>16.55</td>
<td>28.26</td>
</tr>
<tr>
<td>P2</td>
<td>16</td>
<td>18.47</td>
<td>27.38</td>
</tr>
<tr>
<td>P3</td>
<td>16</td>
<td>3.57</td>
<td>17.12</td>
</tr>
<tr>
<td>P4</td>
<td>16</td>
<td>5.24</td>
<td>11.24</td>
</tr>
<tr>
<td>P5</td>
<td>16</td>
<td>6.73</td>
<td>14.27</td>
</tr>
<tr>
<td>P6</td>
<td>16</td>
<td>4.24</td>
<td>23.84</td>
</tr>
<tr>
<td>P7</td>
<td>16</td>
<td>3.45</td>
<td>13.77</td>
</tr>
<tr>
<td>P8</td>
<td>16</td>
<td>17.18</td>
<td>27.51</td>
</tr>
<tr>
<td>P9</td>
<td>16</td>
<td>8.28</td>
<td>19.73</td>
</tr>
<tr>
<td>P10</td>
<td>16</td>
<td>2.48</td>
<td>9.46</td>
</tr>
<tr>
<td>P11</td>
<td>16</td>
<td>20.92</td>
<td>26.48</td>
</tr>
<tr>
<td>P12</td>
<td>16</td>
<td>8.93</td>
<td>15.94</td>
</tr>
<tr>
<td>P13</td>
<td>16</td>
<td>7.83</td>
<td>16.60</td>
</tr>
<tr>
<td>P14</td>
<td>16</td>
<td>15.73</td>
<td>20.52</td>
</tr>
<tr>
<td>Concat</td>
<td>16</td>
<td>14.38</td>
<td>21.67</td>
</tr>
<tr>
<td>Merge</td>
<td>128</td>
<td>38.05</td>
<td>39.91</td>
</tr>
<tr>
<td>DS1</td>
<td>1024</td>
<td>11.65</td>
<td>22.52</td>
</tr>
<tr>
<td>SPC1</td>
<td>4096</td>
<td>9.19</td>
<td>20.00</td>
</tr>
<tr>
<td>MergeS</td>
<td>4096</td>
<td>27.62</td>
<td>40.44</td>
</tr>
</tbody>
</table>