A method and system of managing data retrieval in a computer comprising a cache memory and auxiliary memory comprises organizing pages in the cache memory into a first and second clock list, wherein the first clock list comprises pages with short-term utility and the second clock list comprises pages with long-term utility; requesting retrieval of a particular page in the computer; identifying requested pages located in the cache memory as a cache hit; transferring requested pages located in the auxiliary memory to the first clock list; relocating the transferred requested pages into the second clock list upon achieving at least two consecutive cache hits of the transferred requested page; logging a history of pages evicted from the cache memory; and adaptively varying a proportion of pages marked as short and long-term utility to increase a cache hit ratio of the cache memory by utilizing the logged history of evicted pages.
FIG. 3

INITIALIZE: Set $p = 0$ and set the lists $T_1$, $B_1$, $T_2$, and $B_2$ to empty.

CAR($x$)
INPUT: The requested page $x$.
1: if $(x$ is in $T_1 \cup T_2)$ then /* cache hit */
2: Set the page reference bit for $x$ to one.
3: else /* cache miss */
4: if $(|T_1| + |T_2| = c)$ then /* cache full, replace a page from cache */
   replace()
   /* cache directory replacement */
5: if $(x$ is not in $B_1 \cup B_2)$ and $(|T_1| + |B_1| = c)$ then
6: Discard the LRU page in $B_1$.
7: else if $(|T_1| + |T_2| + |B_1| + |B_2| = 2c)$ and $(x$ is not in $B_1 \cup B_2)$ then
8: Discard the LRU page in $B_2$.
9: endif
10: endif
11: endif
12: /* cache directory miss */
13: if $(x$ is not in $B_1 \cup B_2)$ then
14: Insert $x$ at the tail of $T_1$. Set the page reference bit of $x$ to 0.
15: /* cache directory hit */
16: else if $(x$ is in $B_2)$ then
17: Adapt: Increase the target size for the list $T_1$: $p = \min \{p - \max\{1, |B_1|/|B_1|\}, c\}$
18: Move $x$ at the tail of $T_2$. Set the page reference bit of $x$ to 0.
19: /* cache directory hit */
20: else /* $x$ must be in $B_1$ */
21: Adapt: Decrease the target size for the list $T_1$: $p = \max \{p - \max\{1, |B_1|/|B_1|\}, 0\}$
22: Move $x$ at the tail of $T_2$. Set the page reference bit of $x$ to 0.
23: endif
24: endif

replace()
25: found = 0
26: repeat
27: if $(|T_1| \geq 1, p))$ then
28: if (the page reference bit of head page in $T_1$ is 0) then
29: found = 1;
30: Demote the head page in $T_1$ and make it the MRU page in $B_1$.
31: else
32: Set the page reference bit of head page in $T_1$ to 0, and make it the tail page in $T_5$.
33: endif
34: else
35: if (the page reference bit of head page in $T_2$ is 0) then
36: found = 1;
37: Demote the head page in $T_2$ and make it the MRU page in $B_2$.
38: else
39: Set the page reference bit of head page in $T_2$ to 0, and make it the tail page in $T_5$.
40: endif
41: endif
42: until (found)
<table>
<thead>
<tr>
<th></th>
<th>CAR</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOCK</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LFU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LRU</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Recency</td>
<td>Constant Time</td>
<td>Scan Resistance</td>
<td>Frequency</td>
<td>Lock Contention/ MRU Overhead</td>
</tr>
</tbody>
</table>

**FIG. 4**
FIG. 5

INITIALIZATION: Set p = 0, q = 0, nS = nL = 0, and set the lists T1, B1, T2, and B2 to empty.

CART(x)

1: if (x is in T1 or T2) then /* cache hit */
2: Set the page reference bit for x to one.
3: else /* cache miss */
4: if (|T1| + |T2| = c) then /* cache full, replace a page from cache */
5: replace()
6: if (x \not\in B1 \cup B2) and (|B1| + |B2| = c + 1) and (l(|B1| > \max(0, q)) or (B2 is empty))) then
7: Remove the bottom page in B1 from the history.
8: else if (x \not\in B1 \cup B2) and (|B1| + |B2| = c + 1) then
9: Remove the bottom page in B2 from the history.
10: endif
11: endif
12: /* history miss */
13: if (x is not in B1 \cup B2) then
14: Insert x at the tail of T1. Set the page reference bit of x to 0, set filter bit of x to "S", and nS = nS + 1.
15: /* history hit */
16: else (x is in B1) then
17: Adapt: Increase the target size for the list T1 as p = \min(p + \max(1, nS/|B1|), c). Move x to the tail of T1.
18: Set the page reference bit of x to 0. Set nL = nL + 1. Set type of x to "L".
19: /* history hit */
20: else /* x must be in B2 */
21: Adapt: Decrease the target size for the list T2 as p = \max(p - \max(1, nL/|B2|), 0). Move x to the tail of T2.
22: Set the page reference bit of x to 0. Set nL = nL + 1.
23: if (|T1| + |B1| + |T2| - nS \geq c) then, Set target q = \min(q + 1, 2c - |T1|), endif
24: endif
25: endwhile

replace()
26: while (the page reference bit of the head page in T2 is 1)) then
27: Move the head page in T2 to tail position in T1. Set the page reference bit to 0.
28: if (|T1| + |B1| - nS \geq c) then, Set target q = \min(q + 1, 2c - |T1|), endif
29: endwhile

30: if (the filter bit of the head page in T1 is "L") or (the page reference bit of the head page in T1 is 1))
31: /* The following while loop should stop, if T1 is empty */
32: while (the page reference bit of the head page in T1 is 1))
33: if (the page reference bit of the head page in T1 is 1)
34: Move the head page in T1 to tail position in T1. Set the page reference bit to 0.
35: if (|T1| \geq \min(p + 1, |B1|)) and (the filter bit of the moved page is "S") then,
36: set type of x to "L", nS = nS - 1, and nL = nL + 1.
37: endif
38: else
39: Move the head page in T1 to tail position in T1. Set the page reference bit to 0.
40: Set q = \max(q - 1, c - |T1|).
41: endif
42: endwhile
43: if (|T1| \geq \max(1, p)) then
44: Denote the head page in T1 and make it the MRU page in B1. nS = nS - 1.
45: else
46: Denote the head page in T2 and make it the MRU page in B2. nL = nL - 1.
47: endif
48: end
### FIG. 6

<table>
<thead>
<tr>
<th>Trace Name</th>
<th>Number of Requests</th>
<th>Unique Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>32055473</td>
<td>2311485</td>
</tr>
<tr>
<td>P2</td>
<td>127799495</td>
<td>9133347</td>
</tr>
<tr>
<td>P3</td>
<td>3912296</td>
<td>762543</td>
</tr>
<tr>
<td>P4</td>
<td>19776090</td>
<td>5146832</td>
</tr>
<tr>
<td>P5</td>
<td>229379097</td>
<td>3403835</td>
</tr>
<tr>
<td>P6</td>
<td>12672123</td>
<td>773770</td>
</tr>
<tr>
<td>P7</td>
<td>14521148</td>
<td>1619941</td>
</tr>
<tr>
<td>P8</td>
<td>42243785</td>
<td>977545</td>
</tr>
<tr>
<td>P9</td>
<td>10533489</td>
<td>1369543</td>
</tr>
<tr>
<td>P10</td>
<td>33400528</td>
<td>5679543</td>
</tr>
<tr>
<td>P11</td>
<td>141528425</td>
<td>4579339</td>
</tr>
<tr>
<td>P12</td>
<td>13208930</td>
<td>3153310</td>
</tr>
<tr>
<td>P13</td>
<td>15629738</td>
<td>2497335</td>
</tr>
<tr>
<td>P14</td>
<td>114909968</td>
<td>13814927</td>
</tr>
<tr>
<td>Concat</td>
<td>490139585</td>
<td>47003313</td>
</tr>
<tr>
<td>Merge(P)</td>
<td>490139585</td>
<td>47003313</td>
</tr>
<tr>
<td>DS1</td>
<td>43704979</td>
<td>10516352</td>
</tr>
<tr>
<td>SPC1</td>
<td>41351279</td>
<td>6050636</td>
</tr>
<tr>
<td>S1</td>
<td>3995316</td>
<td>1309698</td>
</tr>
<tr>
<td>S2</td>
<td>17253074</td>
<td>1693344</td>
</tr>
<tr>
<td>S3</td>
<td>16407702</td>
<td>1689882</td>
</tr>
<tr>
<td>Merge (S)</td>
<td>37656092</td>
<td>4692924</td>
</tr>
</tbody>
</table>

### FIG. 7

#### SPC1

<table>
<thead>
<tr>
<th>c (pages)</th>
<th>LRU</th>
<th>CLOCK</th>
<th>ARC</th>
<th>CAR</th>
<th>CART</th>
</tr>
</thead>
<tbody>
<tr>
<td>65536</td>
<td>0.37</td>
<td>0.37</td>
<td>0.82</td>
<td>0.54</td>
<td>0.90</td>
</tr>
<tr>
<td>151072</td>
<td>0.78</td>
<td>0.77</td>
<td>1.62</td>
<td>1.66</td>
<td>1.78</td>
</tr>
<tr>
<td>262144</td>
<td>1.63</td>
<td>1.63</td>
<td>3.23</td>
<td>3.29</td>
<td>3.56</td>
</tr>
<tr>
<td>524288</td>
<td>3.66</td>
<td>3.64</td>
<td>7.56</td>
<td>7.62</td>
<td>8.52</td>
</tr>
<tr>
<td>1048576</td>
<td>9.19</td>
<td>9.31</td>
<td>20.00</td>
<td>20.00</td>
<td>21.90</td>
</tr>
</tbody>
</table>

#### Merge(S)

<table>
<thead>
<tr>
<th>c (pages)</th>
<th>LRU</th>
<th>CLOCK</th>
<th>ARC</th>
<th>CAR</th>
<th>CART</th>
</tr>
</thead>
<tbody>
<tr>
<td>16384</td>
<td>0.20</td>
<td>0.20</td>
<td>1.04</td>
<td>1.03</td>
<td>1.10</td>
</tr>
<tr>
<td>32768</td>
<td>0.40</td>
<td>0.40</td>
<td>2.08</td>
<td>2.07</td>
<td>2.20</td>
</tr>
<tr>
<td>65536</td>
<td>0.79</td>
<td>0.79</td>
<td>4.07</td>
<td>4.05</td>
<td>4.27</td>
</tr>
<tr>
<td>131072</td>
<td>1.59</td>
<td>1.58</td>
<td>7.76</td>
<td>7.76</td>
<td>8.20</td>
</tr>
<tr>
<td>262164</td>
<td>3.23</td>
<td>3.27</td>
<td>14.30</td>
<td>14.25</td>
<td>15.07</td>
</tr>
<tr>
<td>524288</td>
<td>8.06</td>
<td>8.66</td>
<td>24.34</td>
<td>24.47</td>
<td>26.12</td>
</tr>
<tr>
<td>1048576</td>
<td>27.62</td>
<td>29.04</td>
<td>40.44</td>
<td>41.00</td>
<td>41.83</td>
</tr>
<tr>
<td>1572864</td>
<td>50.86</td>
<td>52.24</td>
<td>57.19</td>
<td>57.92</td>
<td>57.64</td>
</tr>
<tr>
<td>2097152</td>
<td>68.68</td>
<td>69.50</td>
<td>71.41</td>
<td>71.71</td>
<td>71.77</td>
</tr>
<tr>
<td>4194304</td>
<td>87.30</td>
<td>87.26</td>
<td>87.26</td>
<td>87.26</td>
<td>87.26</td>
</tr>
<tr>
<td>Workload</td>
<td>c (pages)</td>
<td>space (MB)</td>
<td>LRU CLOCK</td>
<td>ARC</td>
<td>CART</td>
</tr>
<tr>
<td>----------</td>
<td>-----------</td>
<td>------------</td>
<td>------------</td>
<td>-----</td>
<td>-------</td>
</tr>
<tr>
<td>P1</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P2</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P3</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P4</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P5</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P6</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P7</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P8</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P9</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P10</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P11</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P12</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P13</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>P14</td>
<td>37768</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>S1</td>
<td>524288</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>S2</td>
<td>524288</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>S3</td>
<td>524288</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
<tr>
<td>Merge(s)</td>
<td>1043576</td>
<td>16</td>
<td>15.35</td>
<td>1.34</td>
<td>29.83</td>
</tr>
</tbody>
</table>

**FIG. 9**
FIG. 12

Organizing pages in the cache memory into a first clock list and a second clock list, wherein the first clock list comprises pages with short-term utility and the second clock list comprises pages with long-term utility.

Requesting retrieval of a particular page in the computer system.

Identifying requested pages located in the cache memory as a cache hit.

Transferring requested pages located in the auxiliary memory to the first clock list of the cache memory.

Relocating the transferred requested pages into the second clock list upon achieving at least two consecutive cache hits of the transferred requested page.

Logging a history of requested pages transferred from the first clock list into a first list of history pages.

Logging a history of requested pages transferred from the second clock list into a second list of history pages.

Adaptively varying a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory by utilizing the logged history of requested pages.
METHOD AND SYSTEM OF CLOCK WITH ADAPTIVE CACHE REPLACEMENT AND TEMPORAL FILTERING

CROSS-REFERENCE TO RELATED APPLICATION


BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The embodiments of the invention generally relate to cache operations within computer systems, and more particularly to an adaptive cache replacement technique with enhanced temporal filtering in a demand paging environment.

[0004] 2. Description of the Related Art

[0005] Caching is a fundamental problem in computer science. Modern computational infrastructure designs are rich in examples of memory hierarchies where a fast, but expensive main ("cache") memory is placed in front of an inexpensive, but slow auxiliary memory. Caching methodologies manage the contents of the cache so as to improve the overall performance. In particular, caching methodologies are of tremendous interest in databases, virtual memory management, and storage systems, etc., where the cache is RAM and the auxiliary memory is a disk subsystem.

[0006] For simplicity, it is assumed that both the cache and the auxiliary memory are managed in discrete, uniformly-sized units called "pages". If a requested page is present in the cache, then it can be served quickly resulting in a "cache hit". On the other hand, if a requested page is not present in the cache, then it must be retrieved from the auxiliary memory resulting in a "cache miss". Usually, latency on a cache miss is significantly higher than that on a cache hit. Hence, caching methodologies focus on improving the hit ratio. Historically, the assumption of "demand paging" has been used to study caching methodologies. Under demand paging, a page is retrieved from the auxiliary memory to the cache only on a cache miss. In other words, demand paging precludes speculatively pre-fetching pages. Under demand paging, the only question of interest is: When the cache is full, and a new page must be inserted into the cache, which page should be replaced? Currently, the best offline cache replacement policy is the so-called "MIN" technique developed by Laszlo A. Belady, which replaces the page that is used farthest in the future.

[0007] Digital microprocessors use cache memory to hold data likely to be needed in the near future. Cache memory is comparatively fast and is on local memory. Caching usually occurs when data or other instructions are retrieved from the main memory to be used by the microprocessor, and are also stored in the cache. Typically, the cache is constructed from a random access, read/write memory block (RAM), which can access a single stored object, referred to as a line, in a single processor cycle. Preferably, the cache size matches the processor cycle time and is read or written during a given cycle. A server can be configured to receive a stream of requests from clients in a network system to read from or write to a disk drive in the server. These requests form the "workload" for the server.

[0008] Each line in the cache memory contains the data being saved and the address of the data in the main memory (the tag). An example of a simple cache 210 is illustrated in the block diagram of FIG. 1. When the microprocessor makes a reference to the main memory, a part of the reference address, referred to as the index, accesses a single line stored in the cache RAM 212. A “hit” occurs if the tag of the accessed line in the cache 210 matches the reference address of the referenced data. When this happens, the cache RAM 212 immediately supplies the line to the microprocessor. However, a “miss” occurs if the tag of the accessed line in the cache 210 does not match the reference address of the referenced data. When this happens, the address is sent to the main memory to retrieve the requested line. When the main memory sends the line to the microprocessor, it is written into the cache RAM 212 using the same index as the original look-up, along with its tag. However, because the main memory is much slower than the microprocessor, a delay occurs during this retrieval process.

[0009] Additionally, cache memory is used when data is written from a host computer to a long-term data storage device such as a disk drive. Here, data may be written to cache memory in which it is temporarily held with an indication that the data must be written to long-term data storage when the data storage system is able to perform this write operation. When cache memory is used to temporarily delay writing pending data, memory storage locations are removed from the main memory locations generally available to the data storage system in which data may be held pending use by the host.

[0010] Traditionally, under the assumption of demand paging, a cache technique termed the least recently used (LRU) has been used. This popular online policy imitates the MIN technique by replacing the LRU page. When the cache is full, and a page must be demoted to make space for a new page, LRU removes the least recently used page from the cache. The technique LRU is simple to implement, has constant time and space overhead, and it captures "clustered locality of reference" or "recency" property of workloads. However, LRU has three main disadvantages: (i) it does not capture pages with "high frequency" or "long-term-utility"; (ii) it is not resistant to scans which are a sequence of one-time-use-only read/write requests; and (iii) on every hit to a cache page it must be moved to the most recently used (MRU) position. In an asynchronous computing environment where multiple threads may be trying to move pages to the MRU position, the MRU position is protected by a lock to ensure consistency and correctness. This lock typically leads to a great amount of contention, since all cache hits are serialized behind this lock. Such contention is often unacceptable in high performance and high throughput environments such as virtual memory, databases, file systems, and storage controllers.

[0011] Other disadvantages of the LRU technique are that in a virtual memory setting, the overhead of moving a page to the MRU position on every page hit is unacceptable, and while LRU captures the “recency” features of a workload, it does not capture and exploit the “frequency” features of a workload. More generally, if some pages are often re-
requested, but the temporal distance between consecutive requests is larger than the cache size, then LRU cannot take advantage of such pages with "long-term utility". Moreover, LRU can be easily polluted by a scan, that is, by a sequence of one-time use only page requests leading to lower performance.

[0012] Recently, under the assumption of demand paging, a caching technique termed the Adaptive Replacement Cache (ARC) has been used (Nimrod Megiddo and D. S. Modha, ARC: A Self-tuning, Low Overhead Replacement Cache, Proc. 2nd USENIX Conference on File and Storage Technologies (FAST 03), San Francisco, Calif., 115-130, 2003), the complete disclosure of which, in its entirety, is herein incorporated by reference. Comparatively, this caching technique has low computational overhead similar to LRU updating schemes, its space overhead over LRU is negligible, it outperforms LRU for a wide range of workloads and cache sizes, it is self-tuning in that for every workload it dynamically adapts between recency and frequency to increase the hit ratio, and it is scan-resistant, and, hence, avoids cache pollution due to sequential workloads.

[0013] The basic idea behind ARC is that the cache is managed in uniform-sized chunks called "pages". Assuming that the cache can hold c pages, the ARC technique maintains a cache directory that contains 2c pages—c pages in the cache and c history pages. The cache directory of ARC, which is referred to as DBL, maintains two lists: L₁ and L₂. The first list contains pages that have been seen only once recently, while the latter contains pages that have been seen at least twice recently. The replacement technique for managing DBL is: Replace the LRU page in L₁, if |L₁|<c; otherwise, replace the LRU page in L₁. The ARC technique builds on DBL by carefully selecting c pages from the 2c pages in DBL. The basic idea is to divide L₁ into top T₁ and bottom B₁ and to divide L₂ into top T₂ and bottom B₂. The pages in T₁ (resp. T₂) are more recent than those in B₁ (resp. B₂). The methodology sets a target size p for the list T₁. The replacement technique is as follows: Replace the LRU page in T₁, if |T₁|<p; otherwise, replace the LRU page in T₂. The adaptation comes from the fact that the target size p is continuously varied in response to an observed workload. The adaptation rule is as follows: Increase p, if a hit in the history B₁ is observed; similarly, decrease p, if a hit in the history B₁ is observed.

[0014] However, a limitation of ARC is that whenever it observes a hit on a page in L₁∪T₁∪B₁, it immediately promotes the page to L₂=I₂∪B₂ because the page has now been recently seen twice. At an upper level of memory hierarchy, ARC observes two or more successive references to the same page fairly quickly. Such quick successive hits are known as "correlated references" and are not a guarantee of long-term utility of a page, and, hence, such pages pollute L₂, thus reducing system performance. Therefore, there is a need to create a temporal filter that imposes a more stringent test for promotion from I₁ to I₂. Such a temporal filter is of extreme importance in upper levels of memory hierarchy such as file systems, virtual memory, databases, etc.

[0015] The Independent Reference Model (IRM) captures the notion of frequencies of page references. Under the IRM, the requests at different times are stochastically independent. LFU replaces the least frequently used page and is optimal under the IRM but has several potential drawbacks: (i) its running time per request is logarithmic in the cache size; (ii) it is oblivious to recent history; and (iii) it does not adapt well to variable access patterns, wherein it accumulates stale pages with past high frequency counts, which may no longer be useful.

[0016] The last fifteen years have seen development of a number of novel caching methodologies that have attempted to combine "recency" (LRU) and "frequency" (LFU) with the intent of removing one or more disadvantages of LRU. Chronologically, FBR, LRU-2, 2Q, RFU, MQ, and LIRS have been proposed. Unfortunately, each of these techniques poses some prohibitive disadvantages. The ARC technique was introduced to eliminate essentially all of the drawbacks of the above mentioned policies, and further, is self-tuning, requires low overhead, is scan resistant, and has performance characteristics similar to or better than LRU, LFU, FBR, LRU-2, 2Q, MQ, LRFU, and LIRS, even when some of these policies are allowed to select the best, offline values for their tunable parameters, without any need for pre-tuning or user-specified magic parameters. A minor disadvantage of LRU is that it cannot detect loping patterns. This persists in most of the above mentioned cache replacement methodologies, including ARC.

[0017] The CLOCK methodology was developed specifically for low-overhead, low-lock contention environments. Perhaps the oldest methodology along these lines was the First-In First-Out (FIFO) approach that simply maintains a list of all pages in the cache such that head of the list is the oldest arrival and tail of the list is the most recent arrival. However, due to much lower performance than LRU, FIFO in its original form is seldom used today.

[0018] Second chance (SC) is a simple, but extremely effective enhancement to FIFO, where a page reference bit is maintained with each page in the cache while maintaining the pages in a FIFO queue. When a page arrives in the cache, it is appended to the tail of the queue and its reference bit set to zero. Upon a page hit, the page reference bit is set to one. Whenever a page must be replaced, the policy examines the page at the head of the FIFO queue and replaces it if its page reference bit is zero otherwise the page is moved to the tail and its page reference bit is reset to zero. In the latter case, the replacement policy reexamines the new page at the head of the queue, until a replacement candidate with page reference bit of zero is found.

[0019] A key deficiency of SC is that it keeps moving pages from the head of the queue to the tail. This movement makes it somewhat inefficient. CLOCK is functionally identical to SC except that by using a circular queue instead of FIFO it eliminates the need to move a page from the head to the tail. Besides its simplicity, the performance of CLOCK is quite comparable to LRU. CLOCK is a classical cache replacement technique dating back to 1968 that was proposed as a low-complexity approximation to LRU. On every cache hit, the policy LRU needs to move the accessed item to the most recently used position, at which point, to ensure consistency and correctness, it serializes cache hits behind a single global lock. CLOCK eliminates this lock contention, and, hence, can support high concurrency and high throughput environments such as virtual memory and databases. Unfortunately, CLOCK is still plagued by disadvantages of LRU such as disregard for "frequency" and lack of scan-resistance.
A generalized version of CLOCK, namely, GCLOCK, associates a counter with each page that is initialized to a certain value. On a page hit, the counter is incremented. On a page miss, the rotating clock hand sweeps through the clock decrementing counters until a page with a count of zero is found. However, a fundamental disadvantage of GCLOCK is that it requires a counter increment on every page hit which makes it infeasible for virtual memory. There are several other variants of CLOCK, for example, the two-handed clock is used by Solaris® available from Sun Microsystems, Inc., Santa Clara, Calif., USA.

CLOCK maintains a “page reference bit” with every page. When a page is first brought into the cache, its page reference bit is set to zero. The pages in the cache are organized as a circular buffer known as a clock. On a hit to a page, its page reference bit is set to one. Replacement is done by moving a clock hand through the circular buffer. The clock hand can only replace a page with page reference bit set to zero. However, while the clock hand is traversing to find the victim page, if it encounters a page with page reference bit of one, then it resets the bit to zero. Since, on a page hit, there is no need to move the page to the MRU position, no serialization of hits occurs. Moreover, in virtual memory applications, the page reference bit can be turned on by the hardware. Furthermore, the performance of CLOCK is usually quite comparable to LRU.

A recent breakthrough generalization of LRU, namely, Adaptive Replacement Cache (ARC), removes some of the disadvantages associated with LRU. The ARC methodology is scan-resistant, exploits both the recency and the frequency features of the workload in a self-tuning fashion, has low space and time complexity, and outperforms LRU across a wide range of workloads and cache sizes. Furthermore, ARC, which is self-tuning, also has performance characteristics comparable to a number of other state-of-the-art policies even when these policies are allowed the best, offline values for their tunable parameters.

As previously indicated, the CLOCK technique removes some of the disadvantages associated with LRU, while ARC removes other disadvantages associated with LRU. However, there remains a need for a novel page caching technique which removes all of the disadvantages associated with LRU. Thus, while numerous page caching techniques have been developed through the years, there remains a need for an improved page caching technique which overcomes all of the disadvantages of all of the conventional techniques, and which can be implemented easily and efficiently.

SUMMARY OF THE INVENTION

In view of the foregoing, an embodiment of the invention provides a method of managing data retrieval in a computer system comprising a cache memory and an auxiliary memory, wherein the method comprises organizing pages in the cache memory into a first clock list and a second clock list, wherein the first clock list comprises pages with short-term utility and the second clock list comprises pages with long-term utility; requesting retrieval of a particular page in the computer system; identifying requested pages located in the cache memory as a cache hit; transferring requested pages located in the auxiliary memory to the first clock list of the cache memory; relocating the transferred requested pages into the second clock list upon achieving at least two consecutive cache hits of the transferred requested page; logging a history of pages hit from the cache memory; and adaptively varying a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory by utilizing the logged history of evicted pages.

The cache memory is arranged into pages having uniformly-sized units of memory. Also, the process of requesting access to a particular page in the computer system comprises determining whether the particular page is located in the cache memory. The method further comprises maintaining a page reference bit for each page in the cache memory, wherein a new page entering the cache memory comprises a page reference bit of zero, and a page having a cache hit in the cache memory comprises a page reference bit of one.

The method further comprises identifying requested pages located in the auxiliary memory and not in the cache memory as a cache miss, wherein upon identifying the cache miss, the method further comprises evicting a page in either the first clock list or the second clock list if the cache memory is full. Moreover, the method further comprises logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages comprises pages evicted from the first clock list; and a second list of history pages comprises pages evicted from the second clock list.

Furthermore, the method comprises determining whether the requested pages are located in either of the first list of history pages or the second list of history pages; determining whether the cache history is full; evicting a page in the first list of history pages if the transferred requested pages are not located in either of the first list of history pages or the second list of history pages and the cache history is full and a size of the first list of history pages plus a size of the first clock list is equal to a total number of pages in the cache memory; and evicting a page in the second list of history pages if the transferred requested pages are not located in either of the first list of history pages or the second list of history pages and the cache history is full and a size of the first list of history pages plus a size of the first clock list is less than a total number of pages in the cache memory.

Additionally, the method further comprises identifying requested pages located in the auxiliary memory as a cache miss; inserting a transferred requested page at a most recently used (MRU) position in the first clock list; and setting the page reference bit of the transferred requested page to zero. Alternatively, the method further comprises logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; determining whether the requested pages are located in the first list of history pages; establishing a target size of the first clock list; increasing the target size of the first clock list upon a determination that the requested pages are located in the first list of history pages; inserting a transferred requested page at a MRU page position in the second clock list upon a determination that the requested page is located in the first list of history pages; and setting the page reference bit of the transferred requested page to zero.
[0029] Still alternatively, the method further comprises logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; determining whether the requested pages are located in the second list of history pages; establishing a target size of the first clock list; decreasing the target size of the first clock list upon a determination that the requested pages are located in the second list of history pages; inserting a transferred requested page at a MRU page position in the second clock list upon a determination that the requested page is located in the second list of history pages; and setting the page reference bit of the transferred requested page to zero.

[0030] Yet in another embodiment, the method further comprises logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; identifying a least recently used (LRU) page of the first clock list; evicting the LRU page from the first clock list; and transferring the LRU page to a MRU page position in the first list of history pages if the page reference bit of the LRU page is zero and a size of the first clock list is at least as large as a predetermined target size.

[0031] In another embodiment, the method further comprises identifying a LRU page of the first clock list; transferring the LRU page from the first clock list to a MRU page position in the second clock list if the page reference bit of the LRU page is one; and resetting the page reference bit to zero. Alternatively, the method further comprises logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; identifying a LRU page of the second clock list; evicting the LRU page from the second clock list; and transferring the LRU page to a MRU page position in the second list of history pages if the page reference bit of the LRU page is zero and a size of the first clock list is smaller than a predetermined target size. In another alternative embodiment, the method further comprises identifying a LRU page of the second clock list; and transferring the LRU page from the second clock list to a MRU page position in the second clock list if the page reference bit of the LRU page is one.

[0032] Another aspect of the invention provides a program storage device readable by computer, tangibly embodying a program of instructions executable by the computer to perform a method of managing data retrieval in a computer system comprising a cache memory and an auxiliary memory.

[0033] Yet another aspect of the invention provides a system for adaptively managing data retrieval in a computer, wherein the system comprises a cache memory comprising a first clock list and a second clock list, wherein the first clock list comprises pages with short-term utility and the second clock list comprises pages with long-term utility; a query handler adapted to process requests for retrieval of a particular page in the computer; a processor adapted to identify requested pages located in the cache memory as a cache hit; a first data bus adapted to transfer requested pages located in an auxiliary memory of the system to the first clock list of the cache memory; a second data bus adapted to relocate the transferred requested pages into the second clock list upon achieving at least two consecutive cache hits of the transferred requested page; a cache history comprising pages evicted from the cache memory; and a controller adapted to vary a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory by utilizing the logged history of requested pages.

[0034] The cache memory is arranged into pages having uniformly-sized units of memory. The query handler is adapted to determine whether the particular page is located in the cache memory. The system further comprises a bit marker comprising a page reference bit for each page in the cache memory, wherein a new page entering the cache memory comprises a page reference bit of zero, and a page having a cache hit in the cache memory comprises a page reference bit of one. Moreover, the system further comprises a classifier adapted to identify requested pages located in the auxiliary memory as a cache miss, wherein upon identifying the cache miss, the system further comprises a purger adapted to delete a page in either the first clock list or the second clock list if the cache memory is full.

[0035] Additionally, the cache history comprises a first list of history pages comprises pages evicted from the first clock list; and a second list of history pages comprises pages evicted from the second clock list. The system further comprises means for determining whether the requested pages are located in either of the first list of history pages or the second list of history pages; means for determining whether the cache history is full; means for evicting a page in the first list of history pages if the transferred requested pages are not located in either of the first list of history pages or the second list of history pages and the cache history is full and a size of the first list of history pages plus a size of the first clock list is equal to a total number of pages in the cache memory; and means for evicting a page in the second list of history pages if the transferred requested pages are not located in either of the first list of history pages or the second list of history pages and the cache history is full and a size of the first list of history pages plus a size of the first clock list is less than a total number of pages in the cache memory.

[0036] The system further comprises means for identifying requested pages located in the auxiliary memory as a cache miss; means for inserting a transferred requested page at a MRU position in the first clock list; and means for setting the page reference bit of the transferred requested page to zero. Furthermore, the system comprises means for logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; means for determining whether the requested pages are located in the first list of history pages; means for establishing a target size of the first clock list; means for increasing the target size of the first clock list upon a determination that the requested pages are located in the first list of history pages; means for inserting a transferred requested page at a MRU page position in the second clock list upon a determination that the requested page is located in the first list of history pages; and means for setting the page reference bit of the transferred requested page to zero.

[0037] Also, the system comprises means for logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; means for
determining whether the requested pages are located in the second list of history pages; means for establishing a target size of the first clock list; means for decreasing the target size of the first clock list upon a determination that the requested pages are located in the second list of history pages; means for inserting a transferred requested page at a MRU page position in the second clock list upon a determination that the requested page is located in the second list of history pages; and means for setting the page reference bit of the transferred requested page to zero.

[0038] Additionally, the system further comprises means for logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; means for identifying a LRU page of the first clock list; means for evicting the LRU page from the first clock list; and means for transferring the LRU page to a MRU page position in the first list of history pages if the page reference bit of the LRU page is zero and a size of the first clock list is at least as large as a predetermined target size.

[0039] Moreover, the system further comprises means for identifying a LRU page of the first clock list; means for transferring the LRU page from the first clock list to a MRU page position in the second clock list if the page reference bit of the LRU page is one; and means for resetting the page reference bit to zero. The system further comprises means for logging the history of evicted pages into a cache history of the cache memory, wherein the cache history comprises a first list of history pages and a second list of history pages; means for identifying a LRU page of the second clock list; means for evicting the LRU page from the second clock list; and means for transferring the LRU page to a MRU page position in the second list of history pages if the page reference bit of the LRU page is zero and a size of the first clock list is smaller than a predetermined target size. Additionally, the system further comprises means for identifying a LRU page of the second clock list; and means for transferring the LRU page from the second clock list to a MRU page position in the second clock list if the page reference bit of the LRU page is one.

[0040] Another embodiment of the invention provides a system of managing data retrieval in a computer comprising a cache memory and an auxiliary memory, wherein the system comprises means for organizing pages in the cache memory into a first clock list and a second clock list, wherein the first clock list comprises pages with short-term utility and the second clock list comprises pages with long-term utility; means for requesting retrieval of a particular page in the computer system; means for identifying requested pages located in the cache memory as a cache hit; means for transferring requested pages located in the auxiliary memory to the first clock list of the cache memory; means for relocating the transferred requested pages into the second clock list upon achieving at least two consecutive cache hits of the transferred requested page; means for logging a history of pages evicted from the cache memory; and means for adaptively varying a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory by utilizing the logged history of evicted pages.

[0041] The embodiments of the invention provide a novel page caching technique, CLOCK with Adaptive Replacement (CAR), which has several advantages over CLOCK including: (i) it is scan-resistant; (ii) it is self-tuning and it adaptively and dynamically captures the “recency” and “frequency” features of a workload; (iii) it uses essentially the same primitives as CLOCK and, hence, is low-complexity and amenable to a high-concurrency implementation; and (iv) it outperforms CLOCK across a wide-range of cache sizes and workloads. The inventive CAR methodology is derivative of the ARC methodology, and inherits virtually all advantages of ARC including its high performance, but does not serialize cache hits behind a single global lock.

[0042] Additionally, the embodiments of the invention provide another page caching technique, CAR with Temporal filtering (CART), which has all the advantages of CAR, but, in addition, uses a certain temporal filter to distill pages with long-term utility from those with only short-term utility.

[0043] These and other aspects of embodiments of the invention will be better appreciated and understood when considered in conjunction with the following description and the accompanying drawings. It should be understood, however, that the following description, while indicating preferred embodiments of the invention and numerous specific details thereof, is given by way of illustration and not of limitation. Many changes and modifications may be made within the scope of the embodiments of the invention without departing from the spirit thereof, and the invention includes all such modifications.

BRIEF DESCRIPTION OF THE DRAWINGS

[0044] The embodiments of the invention will be better understood from the following detailed description with reference to the drawings, in which:

[0045] FIG. 1 is a schematic diagram of a conventional cache memory system;

[0046] FIG. 2 illustrates a page caching technique according to a first embodiment of the invention;

[0047] FIG. 3 illustrates a page caching methodology corresponding with the page caching technique of FIG. 2 according to a first embodiment of the invention;

[0048] FIG. 4 illustrates a tabular comparison of the conventional page caching techniques and the page caching technique and methodology according to the embodiments of the invention;

[0049] FIG. 5 illustrates a page caching methodology according to a second embodiment of the invention;

[0050] FIG. 6 illustrates an example of disk accesses arranged as traces according to an embodiment of the invention;

[0051] FIG. 7 illustrates a comparison of hit ratios for several page caching techniques;

[0052] FIGS. 8(A) through 8(V) are graphical illustrations of hit ratios achieved by the page caching techniques of FIGS. 2, 3, and 5 according to the embodiments of the invention;

[0053] FIG. 9 illustrates a comparison of hit ratios for various workloads for several page caching techniques;
FIG. 10 is a graphical illustration of an adaptation parameter p versus the virtual time for the page caching technique of FIGS. 2 and 3 according to an embodiment of the invention;

FIG. 11 is a system diagram according to an embodiment of the invention;

FIG. 12 is a flow diagram illustrating a preferred method of an embodiment of the invention; and

FIG. 13 is a computer architecture diagram according to an embodiment of the invention.

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS OF THE INVENTION

The embodiments of the invention and the various features and advantageous details thereof are explained more fully with reference to the non-limiting embodiments that are illustrated in the accompanying drawings and detailed in the following description. It should be noted that the features illustrated in the drawings are not necessarily drawn to scale. Descriptions of well-known components and processing techniques are omitted so as to not unnecessarily obscure the embodiments of the invention. The examples used herein are intended merely to facilitate an understanding of ways in which the embodiments of the invention may be practiced and to further enable those of skill in the art to practice the embodiments of the invention. Accordingly, the examples should not be construed as limiting the scope of the embodiments of the invention.

As mentioned, there remains a need for an improved page caching technique which overcomes all of the disadvantages of all of the conventional techniques, and which can be implemented easily and efficiently. The embodiments of the invention address this need by providing the novel CAR (Clock with Adaptive Replacement) and CART (Clock with Adaptive Replacement and Temporal Filtering) caching techniques. Referring now to the drawings and more particularly to FIGS. 2 through 13 where similar reference characters denote corresponding features consistently throughout the figures, there are shown preferred embodiments of the invention.

FIG. 2 illustrates how the CAR technique provided by the embodiments of the invention functions. The CLOCK lists T1 and T2 contain those pages that are in the cache and the lists B1 and B2 contain history pages that were recently evicted from the cache. The CLOCK list T1 captures “recency” while the CLOCK list T2 captures “frequency.” The lists B1 and B2 are simple LRU lists. Pages evicted from T1 are placed on B1, and those evicted from T2 are placed on B2. The methodology provided by the embodiments of the invention strives to keep B1 to roughly the same size as T1 and B2 to roughly the same size as T2. The algorithm also limits |T1| + |B1| from exceeding the cache size. The sizes of the CLOCK lists T1 and T2 are adapted continuously in response to a varying workload. Whenever a hit in B1 is observed, the target size of T1 is incremented. Similarly, whenever a hit in B2 is observed, the target size of T2 is decremented. The new pages are inserted in either T1 or T2 immediately behind the clock hands, which are shown in FIG. 2 to rotate clockwise.

The page reference bit of new pages is set to zero. Upon a cache hit to any page in T1 U T2, the page reference bit associated with the page is simply set to one. Whenever the T1 clock hand encounters a page with a page reference bit of one, the clock hand moves the page behind the T2 clock hand and resets the page reference bit to zero. Whenever the T1 clock hand encounters a page with a page reference bit of zero, the page is evicted and is placed at the MRU position in B1. Whenever the T2 clock hand encounters a page with a page reference bit of one, the page reference bit is reset to zero. Whenever the T2 clock hand encounters a page with a page reference bit of zero, the page is evicted and is placed at the MRU position in B2.

FIG. 3 illustrates the CAR methodology according to a first embodiment of the invention. Given that c denotes the cache size in pages, the CAR technique maintains four doubly linked lists: T1, T2, B1, and B2. The lists T1 and T2 contain the pages in cache, while the lists B1 and B2 maintain history information about the recently evicted pages. For each page in the cache, that is, in T1 or T2, a page reference bit is maintained that can be set to either one or zero. T1 \u2208 denotes the pages in T1 with a page reference bit of zero and T1 \u2209 denotes the pages in T1 with a page reference bit of one.

The four lists are defined as follows. Each page in T1 \u2208 and each history page in B1 has either been requested exactly once since its most recent removal from T1 U T2 U B1 U B2 or it was requested only once (since inception) and was never removed from T1 U T2 U B1 U B2. Each page in T1 \u2209, each page in T2, and each history page in B2 has either been requested more than once since its most recent removal from T1 U T2 U B1 U B2 or was requested more than once and was never removed from T1 U T2 U B1 U B2.

Intuitively, T1 \u2208 U B1 contains pages that have been seen exactly once recently whereas T1 \u2209 U T2 U B2 contains pages that have been seen at least twice recently. T1 \u2208 U B1 can be thought of as “recency” or “short-term utility” and T1 \u2209 U T2 U B2 as “frequency” or “long-term utility”. In the methodology in FIG. 3, the lists T1 and T2 may be thought of as second chance lists. FIG. 3 depicts T1 and T2 as CLOCK lists. The conventional ARC technique employs a strict LRU ordering on the lists T1 and T2, whereas the inventive CAR uses a one-bit approximation to LRU, that is, SC. The following invariants may be imposed on these lists:

1) 0 ≤ |T1| + |T2| ≤ c.
2) 0 ≤ |T1| + |B1| ≤ c.
3) 0 ≤ |T2| + |B2| ≤ 2c.
4) 0 ≤ |T1| + |T2| + |B1| + |B2| ≤ 2c.
5) If |T1| + |T2| ≤ c, then B1 U B2 is empty.
6) If |T1| + |B1| + |T2| + |B2| ≥ c, then |T1| + |T2| = c.
7) Due to demand paging, once the cache is full, it remains full from then on.

The extra history information contained in lists B1 and B2 are used to guide a continual adaptive process that keeps readjusting the sizes of the lists T1 and T2. For this purpose, a target size p is maintained for the list T1. By implication, the target size for the list T2 is c-p. The extra history leads to a negligible space overhead. The list T1 may contain pages that are marked either as one or zero. Suppose the list T1 is scanned from the head towards the tail until a page marked as zero is encountered. T1 denotes all the pages seen by such a scan, until a page with a page reference bit
of zero is encountered. The cache replacement policy provided by CAR is as follows: If \( T_1 \cup T_2 \) contains \( p \) or more pages, then a page is removed from \( T_1 \), otherwise a page is removed from \( T_1 \cup T_2 \).

\[ \text{[0073]} \] For a better approximation to ARC, the cache replacement policy could be as follows: If \( T_1 \cup T_2 \) contains \( p \) or more pages, then a page is removed from \( T_1 \), otherwise a page is removed from \( T_1 \cup T_2 \). However, this would require maintaining the list \( T_1 \), which seems to entail a much higher overhead on a hit. Hence, the approximate policy is used where \( T_1 \) is used as an approximation to \( T_1 \).

\[ \text{[0074]} \] The cache history replacement policy is as follows: If \( T_1 \cup T_2 \) contains exactly \( c \) pages, then a history page is removed from \( T_1 \), otherwise a history page is removed from \( T_2 \). Once again, for a better approximation to ARC, the cache history replacement policy could be written as: If \( T_1 \cup T_2 \) contains exactly \( c \) pages, then a history page is removed from \( T_1 \), otherwise a history page is removed from \( T_2 \). However, this would require maintaining the size of \( T_1 \), which would require additional processing on a hit, defeating the very purpose of avoiding lock contention.

\[ \text{[0075]} \] With regard to the methodology in FIG. 3, line 1 checks whether there is a hit, and if so, then line 2 simply sets the page reference bit to one. There is no MRU operation akin to LRU or ARC involved. Hence, cache hits are not serialized behind a lock and virtually no overhead is involved. As indicated, the MRU operation is delayed until a replacement must be performed (lines 29 and 36). Line 3 checks for a cache miss, and if so, then line 4 checks if the cache is full, and if so, then line 5 carries out the cache replacement by deleting (evicting) a page from either \( T_1 \) or \( T_2 \). The cache replacement policy “replace( )” is further described below.

\[ \text{[0076]} \] If there is a cache miss (line 3), then lines 6-10 examine whether a cache history has to be replaced. In particular, line 6 indicates that if the requested page is totally new; that is, not in \( B_1 \) or \( B_2 \), and \( T_1 \cup T_2 \) contains, then line 7 indicates that a page in \( B_1 \) is discarded. (line 8) else if the page is totally new and the cache history is completely full, then (line 9) a page in \( B_2 \) is discarded. Finally, if there is a cache miss (line 3), then lines 12-20 carry out movements between the lists and also carry out the adaptation of the target size for \( T_1 \). In particular, (line 12) if the requested page is totally new, then (line 13) it is inserted at the tail of \( T_1 \) and sets its page reference bit to zero, (line 14) else if the requested page is in \( B_1 \), then (line 15) the target size is increased for the list \( T_1 \) and (line 16) the requested page is inserted at the tail of \( T_2 \) and its page reference bit is set to zero. Finally, (line 17) if the requested page is in \( B_2 \), then (line 18) the target size is decreased for the list \( T_1 \) and (line 19) the requested page is inserted at the tail of \( T_2 \) and its page reference bit is set to zero.

\[ \text{[0077]} \] With regard to the cache replacement policy (lines 22-39), the cache replacement policy can only replace a page with a page reference bit of zero. Thus, line 22 declares that no such suitable victim page to replace is yet found, and lines 23-39 keep looping until they find such a page. If the size of the list \( T_1 \) is at least \( p \) and it is not empty (line 24), then the policy examines the head of \( T_1 \) as a replacement candidate. If the page reference bit of the page at the head is zero (line 25), then the desired page (line 26) is found, which is demoted from the cache and is moved to the MRU position in \( B_1 \) (line 27). Else (line 28), if the page reference bit of the LRU page is one, then the page reference bit is reset to one and the page is moved to the MRU end of \( T_2 \) (line 29). Thus, the size of the list \( T_1 \) is effectively increased by one, and the size of the list \( T_1 \) is decreased by zero.

\[ \text{[0078]} \] On the other hand, (line 31) if the size of the list \( T_1 \) is less than \( p \), then the policy examines the page at the head of \( T_1 \) as a replacement candidate. If the page reference bit of the head page is zero (line 32), then the desired page (line 33) is found, and it can be demoted from the cache and can be moved to the MRU position in \( B_2 \) (line 34). Else (line 35), if the page reference bit of the head page is zero, then the page reference bit is reset to zero and the page is moved to the tail of \( T_2 \) (line 36).

\[ \text{[0079]} \] While no MRU operation is needed during a hit, if a page has been accessed and its page reference bit is set to one, then during replacement, such pages will be moved to the MRU end of \( T_2 \) (lines 29 and 36). In other words, CAR approximates ARC by performing a delayed and approximate MRU operation during cache replacement. As implemented, CAR is based on a non-demand-paging framework that uses a free buffer pool of pre-determined size. While cache hits are not serialized, like CLOCK, cache misses are still serialized behind a global lock to ensure correctness and consistency of the lists \( T_1 \), \( T_2 \), \( B_1 \), and \( B_2 \). This miss serialization can be somewhat mitigated by a free buffer pool. FIG. 4 illustrates the advantages of the CAR technique provided by the emodiments of the invention compared with the conventional techniques. As shown, CAR provides a solution for all aspects of caching, namely “recency”, “constant time”, “scan resistance”, “frequency”, and “lock contention/MRU overhead”, which are further described below.

\[ \text{[0080]} \] A limitation of ARC is that two consecutive hits are used as a test to promote a page from “recency” or “short-term utility” to “frequency” or “long-term utility”. At an upper level of memory hierarchy, two or more successive references to the same page are often observed fairly quickly. Such quick successive hits are known as “correlated references” and are typically not a guarantee of long-term utility of pages, and, hence, such pages can cause cache pollution, thus reducing performance. As such, the embodiments of the invention solve this by providing a second embodiment of the invention, namely CLOCK with Adaptive Replacement and Temporal Filtering (CART). The motivation behind CART is to create a temporal filter that imposes a more stringent test for promotion from “short-term utility” to “long-term utility”. The basic idea is to maintain a temporal locality window such that pages that are re-requested within the window are of short-term utility, whereas pages that are re-requested outside the window are of long-term utility. Furthermore, the temporal locality window is itself an adaptable parameter of the methodology.

\[ \text{[0081]} \] Generally, with regard to CART, the idea is to maintain four lists, namely, \( T_1 \), \( T_2 \), \( B_1 \), and \( B_2 \) as before. The pages in \( T_1 \) and \( T_2 \) are in the cache, whereas the pages in \( B_1 \) and \( B_2 \) are in the cache history and are also maintained in a cache directory. However, the pages in \( B_1 \) and \( B_2 \) are in the cache directory, but are not in the cache. For simplicity, it is assumed that \( T_1 \) and \( T_2 \) are implemented as Second Chance lists, but in practice, they would be implemented as CLOCK lists. The following invariants on the sizes of the lists are applicable:
[0082] Generally, the CART methodology provides that the list \( T_1 \) contains exactly \( |T_1| \) least recently used pages (whether or type “S” or “L”) and the list \( T_2 \) contains remaining pages of type “L”. Thus, for CART, \( T_1 \) is a precise representation of “recency” and list \( T_2 \) contains pages of “long-term utility” that LRU would have not kept. The temporal locality window, namely, the size of the list \( T_1 \), is adapted in a workload-dependent, adaptive, online manner. The CART technique decides which list to delete (evict) from according to the rule in lines 36-40 of FIG. 5. A second parameter \( q \) is also maintained, which is the target size for the list \( B_n \). The replacement rule for the cache history is described in lines 6-10 of FIG. 5.

[0083] Letting counters \( n_a \) and \( n_b \) denote the number of pages in the cache that have their filter bit set to “S” and “L”, respectively, it clearly follows that \( 0 \leq n_a + n_b = c \), and once the cache is full, \( n_a + n_b = c \). With regard to the CART methodology of FIG. 5, line 1 checks whether there is a hit, and if so, then line 2 simply sets the page reference bit to one. There is no MRU operation akin to LRU or ARC involved. Hence, cache hits are not serialized behind a lock and virtually no overhead is involved. Moreover, the MRU operation is delayed until a replacement must be performed (lines 24 and 29).

[0084] Line 3 checks for a cache miss, and if so, then line 4 checks if the cache is full, and if so, then line 5 carries out the cache replacement by deleting (evicting) a page from either \( T_1 \) or \( T_2 \). The cache replacement policy “replace” \( \gamma \) is further described below. If there is a cache miss (line 3), then lines 6-10 examine whether a cache history has to be replaced. In particular, (line 6) if the requested page is totally new; that is, not in \( B_1 \) or \( B_2 \) or \( B_1 + B_2 = c+1 \), and \( B_2 \) exceeds its target, then (line 7) a page in \( B_2 \) is discarded, (line 8) else if the page is totally new and the cache history is completely full, then (line 9) a page in \( B_2 \) is discarded.

[0085] Finally, if there is a cache miss (line 3), then lines 11-21 carry out movements between the lists and also carry out the adaptation of the target size for \( T_1 \). In particular, (line 11) if the requested page is totally new, then (line 12) insert it at the tail end of \( T_1 \), its page reference bit is set to zero, the filter bit is set to “S”, the counter \( n_a \) is incremented by 1. As indicated in line 14, else if the requested page is in \( B_1 \), then (line 15) the target size is increased for the list \( T_1 \) (increase the temporal window), and the requested page is inserted at the end of \( T_1 \) and (line 16) its page reference bit is set to zero, and, more importantly, its filter bit is changed to “L”. Finally, (line 17) if the requested page is in \( B_2 \), then (line 18) the target size is decreased for the list \( T_1 \) and the requested page is inserted at the tail end of \( T_1 \), (line 19) its page reference bit is set to zero, and (line 20) the target \( q \) is updated for the list \( B_1 \).

[0086] The essence of the adaptation rule is: on a hit in \( B_1 \), it favors increasing the size of \( T_1 \), and, on a hit in \( B_2 \), it favors decreasing the size of \( T_1 \). As indicated in lines 23-26, while the page reference bit of the head page in \( T_2 \) is 1, then the page is moved to the tail position in \( T_1 \) and the target \( q \) is updated to control the size of \( B_1 \). In other words, these lines capture the movement from \( T_2 \) to \( T_1 \). When this while loop terminates, either \( T_2 \) is empty, or the page reference bit of the head page in \( T_2 \) is set to 0, and, hence, can be removed from the cache if desired.

[0087] Next, as indicated in lines 27-35, while the filter bit of the head page in \( T_1 \) is “L” or the page reference bit of the
head page in \( T_1 \) is 1, these pages are kept moving. When this while loop terminates, either \( T_1 \) will be empty, or the head page in \( T_1 \) has its filter bit set to “S” and the page reference bit is set to 0, and, hence, can be removed from the cache if desired. As provided in lines 28–30 of FIG. 5, if the page reference bit of the head page in \( T_1 \) is 1, then it is made the tail page in \( T_1 \). At the same time, if \( B_j \) is very small or \( T_1 \) is larger than its target, then the temporal filtering constraint is relaxed, and the filter bit is set to “L.” Next, as shown in lines 31–33, if the page reference bit is set to 0, but the filter bit is set to “L,” then the page is moved to the tail position in \( T_2 \). Also, the target \( B_j \) is changed. Lines 36–40 represent the CART cache replacement policy. If \( T_1 \) contains at least \( p \) pages and is not empty, then the LRU page is removed in \( T_1 \), else the LRU page is removed in \( T_2 \).

[0110] The methodologies provided by the embodiments of the invention are validated by experimental analysis. In particular, the several page caching techniques, namely, LRU, CLOCK, ARC, CAR (first embodiment), and CART (second embodiment) are compared. ARC has already been extensively compared to a wide variety of policies such as LRU-2, 2Q, MQ, LIRS, and LRU, and has been shown to be comparable to these policies even when these policies where allowed to employ the best offline selection of their tunable parameters. Hence, ARC is a valid state-of-the-art policy to benchmark against. Finally, since CAR is an approximation to ARC, it of interest to know how much does this approximation cost in terms of performance.

[0111] FIG. 6 summarizes various traces that are used for the purposes of the validation experiments. These traces capture disk accesses by databases, web servers, NT workstations, and a synthetic benchmark for storage controllers. All of the traces have been filtered by upstream caches, and, hence, are representative of workloads seen by storage controllers, disks, or RAID controllers. The traces P1-P14 were collected from workstations running Windows NT (available from Microsoft Corporation, Redmond, Wash., USA) by using Vtrace which captures disk operations through the use of device filters. The traces were gathered over several months. The page size for these traces was 512 bytes. The trace Concat was obtained by concatenating the traces P1-P14. Similarly, the trace Merge(p) was obtained by merging the traces P1-P14 using time stamps on each of the requests. The idea is to synthesize a trace that may resemble a workload seen by a small storage controller.

[0112] The trace DS1 was taken off a database server running at a commercial site running an ERP application on top of a commercial database. The trace is seven days long. SPC1 is a synthetic benchmark trace that contains long sequential scans in addition to random accesses. The page size for this trace was 4 kB. Finally, the three traces S1, S2, and S3 were disk read accesses initiated by a large commercial search engine in response to various web search requests. The trace S1 was captured over a period of one hour, S2 was captured over approximately four hours, and S3 was captured over approximately six hours. The page size for these traces was 4 kB. The trace Merge(S) was obtained by merging the traces S1-S3 using time stamps on each of the requests. The trace Merge(S) is fairly large: 150 GB of requests and a footprint of 18 GB.

[0113] For all traces, only the read requests are considered. All hit ratios reported herein are cold start values, and are indicated as percentages (%). FIG. 7 compares LRU, CLOCK, ARC, CAR, and CART for the traces SPC1 and Merge(S) for various cache sizes. It can be clearly seen that CLOCK has performance characteristics very similar to LRU, and CAR/CART have performance characteristics very similar to ARC. Furthermore, CAR/CART substantially outperforms CLOCK. In FIGS. 8A through 8V (where both the x and y axes use a logarithmic scale), the hit-ratios of CAR are graphically compared to CLOCK for all of the traces P1-P14. As indicated in the plots, the performance of CAR was very close to ARC and CART and the performance of CLOCK was very similar to LRU, and, hence, to avoid clutter, LRU, ARC, and CART are not plotted. It can be clearly seen that across a wide variety of workloads and cache sizes, CAR outperforms CLOCK; sometimes quite dramatically.

[0114] FIG. 9 illustrates a summary of the hit ratios for LRU, CLOCK, ARC, CAR, and CART for various traces and cache sizes. Once again, the same conclusions as above are seen to hold: ARC, CAR, and CART outperform LRU and CLOCK. Moreover, ARC, CAR, and CART have very similar performance characteristics, and CLOCK has performance characteristics very similar to LRU.

[0115] Generally, the embodiments of the invention combine ideas and features from CLOCK and ARC, which prior to the methodologies provided by the invention were impossible to functionally combine, thereby overcoming the disadvantages associated with the LRU technique. Specifically, CAR removes the cache hit serialization problem of LRU and ARC. This property follows since CAR (like CLOCK) does not need to MRU a page on a cache hit. Furthermore, CAR has a very low overhead on cache hits. On a page hit, the only processing that is required is that the associated page reference bit is set to one. This operation can often be performed by the hardware, and, hence, is virtually free. Thus, CAR may be attractive in virtual memory, in high-performance databases, and large storage controllers.

[0116] Additionally, CAR is self-tuning: that is, CAR requires only one tunable parameter \( p \) that balances between recency and frequency. The policy adaptively tunes this parameter in response to an evolving workload so as to increase the hit-ratio. A closer examination of the parameter \( p \) shows that it can fluctuate from recency (\( p=0 \)) to frequency (\( p=\infty \)) and back, all within a single workload. In other words, adaptation is significant. Also, it can be shown that CAR performs as well as its offline counterpart, which is allowed to select the best, offline, fixed value of \( p \) chosen specifically for a given workload and a cache size. In other words, adaptation is significant. FIG. 10 graphically demonstrates how \( p \) fluctuates. Specifically, FIG. 10 is a plot of the adaptation parameter \( p \) (the target size for list \( T_1 \)) versus the virtual time for the CAR methodology. The plot of FIG. 10 is for the P4 and the cache size of 32768 pages. The page size is 512 bytes for both traces. The self-tuning nature of CAR makes it very attractive for deployment in environments where no a priori knowledge of the workloads is available.

[0117] FIG. 11 illustrates a system 100 for adaptively managing data retrieval in a computer 101 according to an embodiment of the invention, wherein the system 100 comprises a cache memory 103 comprising a first clock list \( T_1 \) 107 and a second clock list \( T_2 \) 109, wherein the first clock
list (T_1) 107 comprises pages with short-term utility and the second clock list 109 comprises pages with long-term utility. The system 100 further comprises a query handler 11 adapted to process requests for retrieval of a particular page in the computer 101; a processor 113 adapted to identify requested pages located in the cache memory 109 as a cache hit; a first data bus 115 adapted to transfer requested pages located in an auxiliary memory 105 of the system 100 to the first clock list (T_1) 107 of the cache memory 103; a second data bus 117 adapted to relocate the transferred requested pages into the second clock list (T_2) 109 upon achieving at least two consecutive cache hits of the transferred requested page; a cache history 119 comprising pages evicted from the cache memory 109; and a controller 121 adapted to vary a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory 109 by utilizing the logged history of requested pages. The cache memory 103 is arranged into pages having uniformly-sized units of memory, and the query handler 111 is adapted to determine whether the particular page is located in the cache memory 103.

[0118] The system 100 further comprises a bit marker 123 comprising a page reference bit for each page in the cache memory 103, wherein a new page entering the cache memory 103 comprises a page reference bit of zero, and a page having a cache hit in the cache memory 103 comprises a page reference bit of one. Moreover, the system 100 further comprises a classifier 125 adapted to identify requested pages located in the auxiliary memory 105 as a cache miss, wherein upon identifying the cache miss, the system 100 further comprises a purger 127 adapted to delete (evict) a page in either the first clock list (T_1) 107 or the second clock list (T_2) 109 if the cache memory 103 is full. Furthermore, the cache history 119 comprises a first list of history pages (B_1) 129 and a second list of history pages (B_2) 131.

[0119] The embodiments of the invention achieve several advantages. For example, CAR is scan-resistant, wherein a scan is any sequence of one-time use requests. Such requests will be put on top of the list T_i and will eventually exit from the cache without polluting the high-quality pages in T_j. Moreover, in presence of scans, there will be relatively fewer hits in B_1 as compared to B_2. Hence, the adaptation rule provided by the embodiments of the invention tend to further increase the size of T_2 at the expense of T_1, thus further decreasing the residency time of scan in even T_i.

[0120] Furthermore, CAR provides a high-performance methodology, as it outperforms LRU and CLOCK on a wide variety of traces and cache sizes, and has performance characteristics very comparable to ARC. Moreover, CAR has low space overhead, typically, less than 1% and is simple to implement, as is indicated by the methodology flow given in FIG. 3. Additionally, the CART methodology has all the above advantages of CAR, but, in addition, it employs a much stricter and more precise criterion to distinguish pages with short-term utility from those with long-term utility.

[0121] FIG. 12 illustrates the process flow according to an embodiment of the invention. FIG. 12 illustrates, with reference to FIG. 11, a method of managing data retrieval in a computer system 101 comprising a cache memory 103 and an auxiliary memory 105, wherein the method comprises organizing (201) pages in the cache memory 103 into a first clock list (T_1) 107 and a second clock list (T_2) 109, wherein the first clock list (T_1) 107 comprises pages with short-term utility and the second clock list (T_2) 109 comprises pages with long-term utility; requesting (203) retrieval of a particular page in the computer system 101; identifying (205) requested pages located in the cache memory 103 as a cache hit; transferring (207) requested pages located in the auxiliary memory 105 to the first clock list (T_1) 107 of the cache memory 103; relocating (209) the transferred requested pages into the second clock list (T_2) 109 upon achieving at least two consecutive cache hits of the transferred requested page; logging (211) a history of pages evicted from the first clock list (T_1) into a first list of history pages (B_1) 129; logging (213) a history of pages evicted from the second clock list (T_2) into a second list of history pages (B_2) 131; and adaptively varying (215) a proportion of pages marked as the short-term utility and those marked as the long-term utility to increase a cache hit ratio of the cache memory 103 by utilizing the logged history of evicted pages, wherein the cache memory 103 is arranged into pages having uniformly-sized units of memory, and wherein the process of requesting (203) access to a particular page in the computer system 101 comprises determining whether the particular page is located in the cache memory 103.

[0122] The method further comprises maintaining a page reference bit for each page in the cache memory 103, wherein a new page entering the cache memory 103 comprises a page reference bit of zero, and a page having a cache hit in the cache memory 103 comprises a page reference bit of one. The method further comprises identifying requested pages located in the auxiliary memory 105 and not in the cache memory 103 as a cache miss, wherein upon identifying the cache miss, the method further comprises deleting (evicting) a page in either the first clock list (T_1) 107 or the second clock list (T_2) 109 if the cache memory 103 is full. Moreover, the method further comprises logging the history of evicted pages into a cache history 119 of the cache memory 103, wherein the cache history 119 comprises the first list of history pages (B_1) 129 and the second list of history pages (B_2) 131.

[0123] The method further comprises determining whether the requested pages are located in either of the first list of history pages (B_1) 129 or the second list of history pages (B_2) 131; determining whether the cache history 119 is full; deleting (evicting) a page in the first list of history pages (B_1) 129 if the transferred requested pages are not located in either of the first list of history pages (B_1) 129 or the second list of history pages (B_2) 131 and the cache history 119 is full and a size of the first list of history pages (B_1) 129 plus a size of the first clock list (T_1) 107 is equal to a total number of pages in the cache memory 103; and deleting (evicting) a page in the second list of history pages (B_2) 131 if the transferred requested pages are not located in either of the first list of history pages (B_1) 129 or the second list of history pages (B_2) 131 and the cache history 119 is full and a size of the first list of history pages (B_1) 129 plus a size of the first clock list (T_1) 107 is less than a total number of pages in the cache memory 103.

[0124] Alternatively, the method further comprises identifying requested pages located in the auxiliary memory 105 as a cache miss; inserting a transferred requested page at a MRU position in the first clock list (T_1) 107, and setting the page reference bit of the transferred requested page to zero.
Still alternatively, the method further comprises logging the history of evicted pages into a cache history 119 of the cache memory 103; determining whether the requested pages are located in the first list of history pages (B1) 129; establishing a target size p of the first clock list (T1) 107; increasing the target size p of the first clock list (T1) 107 upon a determination that the requested pages are located in the first list of history pages (B1) 129; inserting a transferred requested page at a MRU page position in the second clock list (T2) 109 upon a determination that the requested page is located in the first list of history pages (B1) 129; and setting the page reference bit of the transferred requested page to zero.

[0125] In another alternate embodiment, the method further comprises determining whether the requested pages are located in the second list of history pages (B2) 131; establishing a target size p of the first clock list (T1) 107; decreasing the target size p of the first clock list (T1) 107 upon a determination that the requested pages are located in the second list of history pages (B2) 131; inserting a transferred requested page at a MRU page position in the second clock list (T2) 109 upon a determination that the requested page is located in the second list of history pages (B2) 131; and setting the page reference bit of the transferred requested page to zero.

[0126] In yet another alternate embodiment, the method further comprises identifying a LRU page of the first clock list (T1) 107; evicting the LRU page from the first clock list (T1) 107; transferring the LRU page to a MRU page position in the first list of history pages (B1) 129 if the page reference bit of the LRU page is zero and a size of the first clock list (T1) 107 is at least as large as a predetermined target size p.

[0127] Still alternatively, the method further comprises transferring the LRU page from the first clock list (T1) 107 to a MRU page position in the second clock list (T2) 109 if the page reference bit of the LRU page is one. In another alternative embodiment, the method further comprises identifying a LRU page of the second clock list (T2) 109; evicting the LRU page from the second clock list (T2) 109; transferring the LRU page to a MRU page position in the second list of history pages (B2) 131 if the page reference bit of the LRU page is zero and a size of the first clock list (T1) 107 is smaller than a predetermined target size p. In another additional alternative embodiment, the method further comprises transferring the LRU page from the second clock list (T2) 109 to a MRU page position in the second clock list (T2) 109 if the page reference bit of the LRU page is one.

[0128] A representative hardware environment for practicing the embodiments of the invention is depicted in FIG. 13. This schematic drawing illustrates a hardware configuration of an information handling/computer system in accordance with the embodiments of the invention. The system comprises at least one processor or central processing unit (CPU) 10. The CPUs 10 are interconnected via system bus 12 to various devices such as a random access memory (RAM) 14, read-only memory (ROM) 16, and an input/output (I/O) adapter 18. The I/O adapter 18 can connect to peripheral devices, such as disk units 11 and tape drives 13, or other program storage devices that are readable by the system. The system can read the inventive instructions on the program storage devices and follow these instructions to execute the methodology of the embodiments of the invention. The system further includes a user interface adapter 19 that connects a keyboard 15, mouse 17, speaker 24, microphone 22, and/or other user interface devices such as a touch screen device (not shown) to the bus 12 to gather user input. Additionally, a communication adapter 20 connects the bus 12 to a data processing network 25, and a display adapter 21 connects the bus 12 to a display device 23 which may be embodied as an output device such as a monitor, printer, or transmitter, for example.

[0129] Generally, the embodiments of the invention provide a caching technique called CLOCK with Adaptive Replacement (CAR), which removes all of the disadvantages associated with the conventional LRU technique. Generally, the inventive CAR technique provided by the embodiments of the invention maintain two clock lists, for example, T1 and T2, where T1 contains pages with “recency” or “short-term utility” and T2 contains pages with “frequency” or “long-term utility”. New pages are first inserted in T1 and graduate to T2 upon passing a certain test of long-term utility. By using a certain precise history mechanism that remembers recently evicted pages from T1 and T2, the embodiments of the invention adaptively determines the sizes of these lists in a data-driven fashion. Using extensive trace-driven simulations, the effectiveness of the methodology provided by the embodiments of the invention is illustrated. Moreover, the trace-driven simulations demonstrate that CAR has performance characteristics comparable to ARC, and substantially outperforms both LRU and CLOCK. Furthermore, like ARC, the CAR methodology is self-tuning and requires no user-specified parameters.

[0130] The CAR methodology considers two consecutive hits to a page as a test of its long-term utility. At upper levels of memory hierarchy, for example, virtual memory, databases, and file systems, two or more successive references to the same page are observed fairly quickly. Furthermore, the embodiments of the invention provide a second page caching technique called CAR with Temporal filtering (CART), which has all of the advantages of CAR, but, imposes a more stringent test to demarcate between pages with long-term utility from those with only short-term utility. The CART technique may be more suitable for disk, RAID, and storage controllers, whereas the CART technique may be more suited to virtual memory, databases, and file systems.

[0131] The foregoing description of the specific embodiments will so fully reveal the general nature of the invention that others can, by applying current knowledge, readily modify and/or adapt for various applications such specific embodiments without departing from the generic concept, and, therefore, such adaptations and modifications should and are intended to be comprehended within the meaning and range of equivalents of the disclosed embodiments. It is to be understood that the phraseology or terminology employed herein is for the purpose of description and not of limitation. Therefore, while the invention has been described in terms of preferred embodiments, those skilled in the art will recognize that the invention can be practiced with modification within the spirit and scope of the appended claims.

What is claimed is:

1. A method of managing data retrieval in a computer system comprising a cache memory and an auxiliary memory, said method comprising:
organizing pages in said cache memory into a first clock list and a second clock list, wherein said first clock list comprises pages with short-term utility and said second clock list comprises pages with long-term utility;

requesting retrieval of a particular page in said computer system;

identifying requested pages located in said cache memory as a cache hit;

transferring requested pages located in said auxiliary memory to said first clock list of said cache memory;

relocuting the transferred requested pages into said second clock list upon achieving at least two consecutive cache hits of said transferred requested page;

logging a history of pages evicted from said cache memory; and

adaptively varying a proportion of pages marked as said short-term utility and those marked as said long-term utility to increase a cache hit ratio of said cache memory by utilizing the logged history of evicted pages.

2. The method of claim 1, wherein said cache memory is arranged into pages having uniformly-sized units of memory.

3. The method of claim 1, wherein said requesting access to a particular page in said computer system comprises determining whether said particular page is located in said cache memory.

4. The method of claim 1, further comprising maintaining a page reference bit for each page in said cache memory, wherein a new page entering said cache memory comprises a page reference bit of zero, and a page having a cache hit in said cache memory comprises a page reference bit of one.

5. The method of claim 1, further comprising identifying requested pages located in said auxiliary memory and not in said cache memory as a cache miss.

6. The method of claim 5, wherein upon identifying said cache miss, said method further comprises evicting a page in either said first clock list or said second clock list if said cache memory is full.

7. The method of claim 6, further comprising logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages comprising pages evicted from said first clock list; and a second list of history pages comprising pages evicted from said second clock list.

8. The method of claim 7, further comprising:

determining whether said requested pages are located in either of said first list of history pages or said second list of history pages;

determining whether said cache history is full;

 evaciting a page in said first list of history pages if said transferred requested pages are not located in either of said first list of history pages or said second list of history pages and said cache history is full and a size of said first list of history pages plus a size of said first clock list is equal to a total number of pages in said cache memory; and

evicting a page in said second list of history pages if said transferred requested pages are not located in either of said first list of history pages or said second list of history pages and said cache history is full and a size of said first list of history pages plus a size of said first clock list is less than a total number of pages in said cache memory.

9. The method of claim 4, further comprising:

identifying requested pages located in said auxiliary memory as a cache miss;

inserting a transferred requested page at a most recently used (MRU) position in said first clock list; and

setting said page reference bit of said transferred requested page to zero.

10. The method of claim 4, further comprising:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

determining whether said requested pages are located in said first list of history pages;

establishing a target size of said first clock list;

increasing said target size of said first clock list upon a determination that said requested pages are located in said first list of history pages;

inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said first list of history pages; and

setting said page reference bit of said transferred requested page to zero.

11. The method of claim 4, further comprising:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

determining whether said requested pages are located in said second list of history pages;

establishing a target size of said first clock list;

decreasing said target size of said first clock list upon a determination that said requested pages are located in said second list of history pages;

inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said second list of history pages; and

setting said page reference bit of said transferred requested page to zero.

12. The method of claim 4, further comprising:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

identifying a least recently used (LRU) page of said first clock list;
evicting said LRU page from said first clock list; and
transferring said LRU page to a most recently used
(MRU) page position in said second list of history pages
if said page reference bit of said LRU page is zero and
a size of said first clock list is at least as large as a
predetermined target size.
13. The method of claim 4, further comprising:
identifying a least recently used (LRU) page of said first
clock list;
transferring said LRU page from said first clock list to a
most recently used (MRU) page position in said second
clock list if said page reference bit of said LRU page is
one; and
resetting said page reference bit to zero.
14. The method of claim 4, further comprising:
logging said history of evicted pages into a cache history
of said cache memory, wherein said cache history
comprises a first list of history pages and a second list
of history pages;
identifying a least recently used (LRU) page of said second
clock list;
evicting said LRU page from said second clock list; and
transferring said LRU page to a most recently used
(MRU) page position in said second list of history pages
if said page reference bit of said LRU page is
zero and a size of said first clock list is smaller than a
predetermined target size.
15. The method of claim 4, further comprising:
identifying a least recently used (LRU) page of said second
clock list; and
transferring said LRU page from said second clock list to
a most recently used (MRU) page position in said second
clock list if said page reference bit of said LRU page is
one.
16. A program storage device readable by computer,
tangibly embodying a program of instructions executable by
said computer to perform a method of managing data
retrieval in a computer system comprising a cache memory
and an auxiliary memory, said method comprising:
organizing pages in said cache memory into a first clock
list and a second clock list, wherein said first clock list
comprises pages with short-term utility and said second
clock list comprises pages with long-term utility;
requesting retrieval of a particular page in said computer
system;
identifying requested pages located in said cache memory
as a cache hit;
transferring requested pages located in said auxiliary
memory to said first clock list of said cache memory;
relocating the transferred requested pages into said second
clock list upon achieving at least two consecutive cache
hits of said transferred requested page;
logging a history of pages evicted from said cache
memory; and
adaptively varying a proportion of pages marked as said
short-term utility and those marked as said long-term
utility to increase a cache hit ratio of said cache
memory by utilizing the logged history of evicted
pages.
17. The program storage device of claim 16, wherein said
cache memory is arranged into pages having uniformly-
sized units of memory.
18. The program storage device of claim 16, wherein said
requesting access to a particular page in said computer
system comprises determining whether said particular page
is located in said cache memory.
19. The program storage device of claim 16, wherein said
method further comprises maintaining a page reference bit
for each page in said cache memory, wherein a new page
entering said cache memory comprises a page reference bit
of zero, and a page having a cache hit in said cache memory
comprises a page reference bit of one.
20. The program storage device of claim 16, wherein said
method further comprises identifying requested pages
located in said auxiliary memory and not in said cache
memory as a cache miss.
21. The program storage device of claim 20, wherein upon
identifying said cache miss, said method further comprises
evicting a page in either said first clock list or said second
clock list if said cache memory is full.
22. The program storage device of claim 21, wherein said
method further comprises logging said history of evicted
pages into a cache history of said cache memory, wherein
said cache history comprises a first list of history pages
comprising pages evicted from said first clock list; and a
second list of history pages comprising pages evicted from
said second clock list.
23. The program storage device of claim 22, wherein said
method further comprises:
determining whether said requested pages are located in
either of said first list of history pages or said second
list of history pages;
determining whether said cache history is full;
evicting a page in said first list of history pages if said
transferred requested pages are not located in either of
said first list of history pages or said second list of
history pages and said cache history is full and a size of
said first list of history pages plus a size of said first
clock list is equal to a total number of pages in said
cache memory; and
evicting a page in said second list of history pages if said
transferred requested pages are not located in either of
said first list of history pages or said second list of
history pages and said cache history is full and a size of
said first list of history pages plus a size of said first
clock list is less than a total number of pages in said
cache memory.
24. The program storage device of claim 19, wherein said
method further comprises:
identifying requested pages located in said auxiliary
memory as a cache miss;
inserting a transferred requested page at a most recently
used (MRU) position in said first clock list; and
setting said page reference bit of said transferred
requested page to zero.
25. The program storage device of claim 19, wherein said
method further comprises:
logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

determining whether said requested pages are located in said first list of history pages;

establishing a target size of said first clock list;

increasing said target size of said first clock list upon a determination that said requested pages are located in said first list of history pages;

inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said first list of history pages; and

setting said page reference bit of said transferred requested page to zero.

26. The program storage device of claim 19, wherein said method further comprises:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

determining whether said requested pages are located in said second list of history pages;

establishing a target size of said first clock list;

decreasing said target size of said first clock list upon a determination that said requested pages are located in said second list of history pages;

inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said second list of history pages; and

setting said page reference bit of said transferred requested page to zero.

27. The program storage device of claim 19, wherein said method further comprises:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

identifying a least recently used (LRU) page of said first clock list;

evicting said LRU page from said first clock list; and

transferring said LRU page to a most recently used (MRU) page position in said first list of history pages if said page reference bit of said LRU page is zero and a size of said first clock list is smaller than a predetermined target size.

28. The program storage device of claim 19, wherein said method further comprises:

identifying a least recently used (LRU) page of said first clock list;

transferring said LRU page from said first clock list to a most recently used (MRU) page position in said second clock list if said page reference bit of said LRU page is one; and

resetting said page reference bit to zero.

29. The program storage device of claim 19, wherein said method further comprises:

logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;

identifying a least recently used (LRU) page of said second clock list;

evicting said LRU page from said second clock list; and

transferring said LRU page to a most recently used (MRU) page position in said second list of history pages if said page reference bit of said LRU page is zero and a size of said first clock list is smaller than a predetermined target size.

30. The program storage device of claim 19, wherein said method further comprises:

identifying a least recently used (LRU) page of said second clock list; and

transferring said LRU page from said second clock list to a most recently used (MRU) page position in said second list of history pages if said page reference bit of said LRU page is one.

31. A system for adaptively managing data retrieval in a computer, said system comprising:

a cache memory comprising a first clock list and a second clock list, wherein said first clock list comprises pages with short-term utility and said second clock list comprises pages with long-term utility;

a query handler adapted to process requests for retrieval of a particular page in said computer;

a processor adapted to identify requested pages located in said cache memory as a cache hit;

a first data bus adapted to transfer requested pages located in an auxiliary memory of said system to said first clock list of said cache memory;

a second data bus adapted to relocate the transferred requested pages into said second clock list upon achieving at least two consecutive cache hits of said transferred requested page;

a cache history comprising pages evicted from said cache memory; and

a controller adapted to vary a proportion of pages marked as said short-term utility and those marked as said long-term utility to increase a cache hit ratio of said cache memory by utilizing the logged history of requested pages.

32. The system of claim 31, wherein said cache memory is arranged into pages having uniformly-sized units of memory.

33. The system of claim 31, wherein said query handler is adapted to determine whether said particular page is located in said cache memory.

34. The system of claim 31, further comprising a bit marker comprising a page reference bit for each page in said cache memory, wherein a new page entering said cache...
memory comprises a page reference bit of zero, and a page having a cache hit in said cache memory comprises a page reference bit of one.

35. The system of claim 31, further comprising a classifier adapted to identify requested pages located in said auxiliary memory as a cache miss.

36. The system of claim 35, wherein upon identifying said cache miss, said system further comprises a purger adapted to delete a page in either said first clock list or said second clock list if said cache memory is full.

37. The system of claim 35, wherein said cache history comprises a first list of history pages comprising pages evicted from said first clock list; and a second list of history pages comprising pages evicted from said second clock list.

38. The system of claim 37, further comprising:

- means for determining whether said requested pages are located in either of said first list of history pages or said second list of history pages;
- means for determining whether said cache history is full;
- means for evicting a page in said first list of history pages if said transferred requested pages are not located in either of said first list of history pages or said second list of history pages and said cache history is full and a size of said first list of history pages plus a size of said first clock list is equal to a total number of pages in said cache memory; and
- means for evicting a page in said second list of history pages if said transferred requested pages are not located in either of said first list of history pages or said second list of history pages and said cache history is full and a size of said first list of history pages plus a size of said first clock list is less than a total number of pages in said cache memory.

39. The system of claim 34, further comprising:

- means for identifying requested pages located in said auxiliary memory as a cache miss;
- means for inserting a transferred requested page at a most recently used (MRU) position in said first clock list; and
- means for setting said page reference bit of said transferred requested page to zero.

40. The system of claim 34, further comprising:

- means for logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;
- means for determining whether said requested pages are located in said first list of history pages;
- means for establishing a target size of said first clock list;
- means for increasing said target size of said first clock list upon a determination that said requested pages are located in said first list of history pages;
- means for inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said first list of history pages; and
- means for setting said page reference bit of said transferred requested page to zero.

41. The system of claim 34, further comprising:

- means for logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;
- means for determining whether said requested pages are located in said second list of history pages;
- means for establishing a target size of said first clock list;
- means for decreasing said target size of said first clock list upon a determination that said requested pages are located in said second list of history pages;
- means for inserting a transferred requested page at a most recently used (MRU) page position in said second clock list upon a determination that said requested page is located in said second list of history pages; and
- means for setting said page reference bit of said transferred requested page to zero.

42. The system of claim 34, further comprising:

- means for logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;
- means for identifying a least recently used (LRU) page of said first clock list;
- means for evicting said LRU page from said first clock list; and
- means for transferring said LRU page to a most recently used (MRU) page position in said first list of history pages if said page reference bit of said LRU page is zero and a size of said first clock list is at least as large as a predetermined target size.

43. The system of claim 34, further comprising:

- means for identifying a least recently used (LRU) page of said first clock list;
- means for transferring said LRU page from said first clock list to a most recently used (MRU) page position in said second clock list if said page reference bit of said LRU page is one; and
- means for resetting said page reference bit to zero.

44. The system of claim 34, further comprising:

- means for logging said history of evicted pages into a cache history of said cache memory, wherein said cache history comprises a first list of history pages and a second list of history pages;
- means for identifying a least recently used (LRU) page of said second clock list;
- means for evicting said LRU page from said second clock list; and
- means for transferring said LRU page to a most recently used (MRU) page position in said second list of history pages if said page reference bit of said LRU page is zero and a size of said first clock list is smaller than a predetermined target size.
45. The system of claim 34, further comprising:
means for identifying a least recently used (LRU) page of said second clock list; and
means for transferring said LRU page from said second clock list to a most recently used (MRU) page position in said second clock list if said page reference bit of said LRU page is one.
46. A system of managing data retrieval in a computer comprising a cache memory and an auxiliary memory, said system comprising:
means for organizing pages in said cache memory into a first clock list and a second clock list, wherein said first clock list comprises pages with short-term utility and said second clock list comprises pages with long-term utility;
means for requesting retrieval of a particular page in said computer system;
means for identifying requested pages located in said cache memory as a cache hit;
means for transferring requested pages located in said auxiliary memory to said first clock list of said cache memory;
means for relocating the transferred requested pages into said second clock list upon achieving at least two consecutive cache hits of said transferred requested page;
means for logging a history of pages evicted from said cache memory; and
means for adaptively varying a proportion of pages marked as said short-term utility and those marked as said long-term utility to increase a cache hit ratio of said cache memory by utilizing the logged history of evicted pages.

* * * * *