The Sisyphus Database Retrieval Software Performance Antipattern

Third International Workshop on Software and Performance
July 24-26, 2002
Rome, Italy

Robert F. Dugan Jr.
Dept. of Computer Science
Stonehill College
Easton, MA 02357 USA
bdugan@stonehill.edu

Ephraim P. Glinert
Dept. of Computer Science
Rensselaer Polytechnic Inst.
Troy, NY 12180 USA
glinert@cs.rpi.edu

Ali Shokoufandeh
Dept. of Math/Computer Science
Drexel University
Philadelphia, PA 19104 USA
ashokouf@mcs.drexel.edu
Overview

- Software Performance Antipatterns
- Sisyphus Database Retrieval Antipattern
- Solutions
- Experiments
- Real World Challenges
- Future Work
Software Performance Antipatterns

Software Design Patterns:
- Effective solution to a common software design problem
- singleton, proxy, iterator, observer/listener
  [Gamma et al. 1995]

Software Design Antipatterns:
- “A commonly occurring solution to a problem that generates decidedly negative consequences.”
  [Brown et al. 1998]
- “god” class, dead code, class proliferation
Software Performance Antipatterns

- “Software Performance Antipatterns”, Smith and Williams, WOSP 2000
  - “God” Class
  - Circuitous Treasure Hunt
  - Excessive Dynamic Allocation
  - One Lane Bridge

- A commonly occurring solution to a software design problem that generates decidedly negative performance consequences
Sisyphus Database Retrieval Antipattern

1) Issue request to display list subset
2) Issue database query to retrieve entire list
3) Return query results
4) Determine number of items displayed
5) Iterate through result set discarding all items until first item to display is reached
6) Continue through result set rendering items for display until last item to display is reached
7) Discard remaining result set
8) Display subset

examples: email, address book, search results
Sisyphus Database Retrieval Antipattern

- Key to this antipattern is the processing necessary to retrieve the entire list from which a subset is extracted must be repeated.

- Recalls Greek myth of Sisyphus damned for all eternity to push a stone up a hill only to watch it roll back down again.
Sisyphus Database Retrieval Antipattern

- Three tier system selected.
- SPE techniques used to model and analyze antipattern.

<table>
<thead>
<tr>
<th>Resource</th>
<th>Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disk</td>
<td><strong>Database Server</strong>: Number of disk I/O operations rises linearly with the size of the total list. I/O reduction possible with database caching, but memory resource contention as system scales to more users</td>
</tr>
</tbody>
</table>
| CPU           | **Browser**: linear dependence on list subset  
                 **Web Server**: linear dependence on start position of subset within result set; linear dependence on list subset  
                 **Database Server**: log linearly with the size of the total list; linear dependence on start position of subset within result set; linear dependence on list subset |
| Network       | **Browser-Web Server**: linear dependence on list subset  
                 **Web-Database Server**: linear dependence on start position of subset within result set |
Solutions: Index and Rownum

- multi-attribute index and rownum

  ```sql
  select lname, fname, phone, address
  from contacts
  where userid=45 and rownum <= 50
  ```

- Advantages:
  - processing beyond subset eliminated
  - sorting result set eliminated

- Disadvantages
  - linear dependence on subset start position
  - multi-attribute index prevents dynamic sorting
  - no total list size
Solutions: Upper/Lower Bound

- multi-attribute index, lower bound attribute value, rownum
  
  select lname, fname, phone, address
  from contacts
  where userid=45 and rownum <= SUBSETSIZE
  and lname > ENDSUBSETLASTNAME

- Linear dependence on list subset size

- Disadvantages:
  - lower bound attribute must be unique
  - multi-attribute index prevents dynamic sorting
  - no total list size
Solutions: Sequence Numbers

- Each list element assigned unique sequence number
- Combination of user and sequence number is unique
  ```sql
  select lname, fname, phone, address
  from contacts
  where userid=45 and lnameSeq >= subListStart
  and lnameSeq <= subListEnd
  ```

Advantages
- Linear dependence on list subset size
- No restriction on duplicate list elements
- Trivial to compute list size
- Multiple sorting criteria possible
- Cost of maintaining sequence number
Solutions: Caching

- Amortize cost of full list retrieval across subset views
- List resides outside database after first subset retrieval
- Advantages:
  - Useful when \( \text{listSize/subSetViews} \leq \text{subListSize} \), e.g. list shared across multiple users
  - Resources eliminated completely after first retrieval
  - Linear dependence on list subset size
  - Compute total list size once
- Disadvantages
  - Potentially significant response time for first retrieval
  - Cache state maintained between requests complicating scaling
  - Cache consistency
  - Tier memory required for cache
## Experiments

<table>
<thead>
<tr>
<th>Subset Start</th>
<th>Antipattern get/discard tuple (ms)</th>
<th>Antipattern get/render tuple (ms)</th>
<th>Seq. Number get/discard tuple (ms)</th>
<th>Seq. Number get/render tuple (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.26</td>
<td>35.22</td>
<td>---</td>
<td>20.93</td>
</tr>
<tr>
<td>20</td>
<td>31.75</td>
<td>19.99</td>
<td>---</td>
<td>21.05</td>
</tr>
<tr>
<td>40</td>
<td>48.32</td>
<td>22.06</td>
<td>---</td>
<td>21.43</td>
</tr>
<tr>
<td>80</td>
<td>81.02</td>
<td>20.11</td>
<td>---</td>
<td>22.01</td>
</tr>
<tr>
<td>160</td>
<td>144.39</td>
<td>19.96</td>
<td>---</td>
<td>23.14</td>
</tr>
<tr>
<td>320</td>
<td>278.29</td>
<td>21.1</td>
<td>---</td>
<td>25.33</td>
</tr>
<tr>
<td>640</td>
<td>548.97</td>
<td>20.78</td>
<td>---</td>
<td>30.13</td>
</tr>
<tr>
<td>1280</td>
<td>1057.9</td>
<td>20.29</td>
<td>---</td>
<td>39.33</td>
</tr>
</tbody>
</table>
Real World Challenges

- eCal provides a web based calendar/address book system
- Antipattern uncovered by performance engineering
- Resistance to design change from database and application development teams because of schedules
- Experimental evidence reinforced antipattern as problem for lists above 100 elements
- Debate over average list size per user
- List subset handling logic encapsulated in stored procedures isolating application logic
- Monitor average list sizes in production, when average exceeds 100, then sequence number solution used
Future Work

- Software Performance Antipattern Workshop
  - Great opportunity for veteran performance engineers from industry to contribute
  - Compendium of Antipatterns much like Addison-Wesley’s Design Patterns book
  - Coming soon (WOSP 2003?, SIGMETRICs?)

- Caching Techniques