SCHOOL OF PHYSICS, GEORGIA INSTITUTE OF TECHNOLOGY

RPC-Based Proxy Server

Group member :

Xiong Ding 902934749
Yubin Zhou

March 23, 2015

1 INTRODUCTION

2 CACHE DESIGN

Cache is implemented by the combination of **Hash table** and **double linked list**. Each hash entry is a tuple **<URL, node*>**, where the node pointer points to a node in the double linked list (cache list). Each node is a structure containing the URL and the corresponding web page content.

We use the C++ STL container **std::unordered_map** to hold the hash table. The searching operation in the unordered_map has average complexity $O(1)$, and the worst case is $O(N)$. Double linked list is used to hold the cached web pages. We insert new nodes at the head, and remove nodes at the tail. Different cache policy will implement different rules to manipulate this list, and insertion and deletion operation has complexity $O(1)$.

The cache class can support two different operations: **get()** and **put()**. whenever there comes a URL, **get()** will try to look up the hash table, if the corresponding entry is found, then the web page content is returned , and also **get()** may modify the node list to reflect recent access (only for LRU policy). On the other hand, **put()** function is responsible for inserting new node to list and delete some nodes if the total size cached goes beyond the limits. Therefore, different cache policies only need to implement different **get()** and **put()**, such that the interface keeps uniform.

3 CACHE POLICIES

3.1 LRU

The Least Recent Used policy will keep track of the access order of cache list. In our implementation, the closer to the list head, the fresher of the web page contained in this node. So every time there is a cache hit, the corresponding cache node will be moved to the head, and if the cache is full, the nodes closest to the tail will first be removed.

- **Pros:** LRU can enhance locality. Recently visited websites are likely to be visited again.
- **Cons:** Performance issue: every lookup of the cache needs to update the double linked cache list no matter whether it is a hit of miss.

3.2 RANDOM

Random policy evicts the cache entry randomly When the total cache size excludes the maximal size. It can be implemented by a random number generator to choose the replace cache entry.

- **Pros:** It won't result in extremely bad performance. for some special patterns, because the eviction decision is not based on the access pattern; thus, it has similar performance for all kinds of patterns.
- **Cons:** It has higher rates compared with other cache policies which take account of access pattern like LRU, because users are likely to browse a few web pages repeatedly.

3.3 FIFO

The "first in first out" policy will evicts the entry that was put into the cache earliest. Cache entries are always inserted at the tail of the cache container, and evicted from the head of the cache container.

- **Pros:** Usually, the oldest entry in the cache is least likely to be accessed again. FIFO reflects the "age" of each cache entry, and evicts the oldest page, so it has relatively good performance compared to random policy.
- **Cons:** In some cases, the oldest cache may be accessed again, such as browsing a few web pages in a circular manner, so there is a large number of cache misses.

4 EVALUATION METRICS

5 WORKLOADS GENERATION

6 EXPERIMENT DESCRIPTION

7 EXPERIMENTAL RESULTS & ANALYSIS

8 CONCLUSION