CS C341 / IS C361
Data Structures & Algorithms

Dictionary Data Structures - Hashing

Bloom Filters

- Motivation
- Implementation
- Analysis
- General Scenario
- Applications.

Las Vegas vs. Monte Carlo Techniques

10/12/2011 Sundar B. CSIS, BITS, Pilani

Bloom Filters - Motivation

- Example Problem: Stemming of words (as part of indexing documents in a search engine)
- Consider this outline for stemming :

```
for each word w
    if (w is an exception word)
   then   return getStem(w,D)
   else  return apply-simple-rule(w)
```

Need dictionary lookup on disk

- Cost for checking exceptions:
  - N * Td   where
    - N is # words and
    - Td is lookup time (on disk)

Bloom Filters - Motivation

- Suppose we can trade-off space for false positives (in lookup):

```
for each word w
    if (w is in Dm )          // in-memory lookup (probabilistic)
    then { s = getStem(w, Dd)      // disk lookup (deterministic)
      if invalid(s) then return apply-simple-rule(w); else return s;
    } else { return apply-simple-rule(w); }
```

- Cost for checking exceptions:
  - N * Tm + (r + f)*N*Td
    - r is the proportion of exception words
    - f is false positive rate
    - Tm is lookup time in memory
    - Td is lookup time on disk
  - Time Saved: (1 – r –f) * (Td – Tm) / Td

Bloom Filters – An implementation

- Hash table is an array of bits indexed from 0 to m-1.
  - Initialize all bits to 0.
  - insert(k):
    - Compute $h_1(k), h_2(k), \ldots, h_d(k)$ where each h_i is a hash function resulting in one of the m addresses.
    - Set all those addressed locations to 1.
  - find(k):
    - Compute $h_1(k), h_2(k), \ldots, h_d(k)$
    - If all addressed locations are 1 then k is **found**

      Else k is **not found**

Always correct.

Not necessarily correct!

Bloom Filters - analysis

- Consider a table H of size m.
- Assume we use d “good” hash functions.
- After n elements have been inserted, the probability that *a specific location is 0* is given by
  - $p = (1 - 1/m)^{dn} \approx e^{-dn/m}$ // Why?
- Let q be the proportion of 0 bits after insertion of n elements
  - Then the expected value E(q) = p
- Claim (w/o proof):
  - With high probability q is close to its mean.
- So, the false positive rate is:
  - $f = (1-q)^d = (1-p)^d = (1 - e^{-dn/m})^d$ // Why?

Bloom Filters - Summary

- A Bloom Filter is a probabilistic data structure:
  - If a value is not found then it is definitely not a member
  - If a value is found then it may or may not be a member.
- The error probability can be traded for space.
  - In practice, one can get low error probability with a (small) constant number of bits per element: (1 in our example implementation) .
- In general, whenever a large dictionary (or set) has to be stored on disk (or remotely on the network)
  - then a probabilistic data structure can be stored in memory and used as a filter thereby limiting the queries on disk (or over the network).

Bloom Filters – Applications

1. Dictionaries (for spell-checkers, passwords, etc.)
2. Distributed Databases – exchange Bloom Filters instead of full lists.
   - A distributed database may split its data onto multiple computers over a network.
   - When a query involving data from multiple data sets is to be executed
     - then you may need to send an entire table from one computer to another
     - Alternatively one can send a Bloom filter, after which a subset of the data can be queried over the network.

Bloom Filters – Applications [2]

3. Network Caches

- Multiple clients on the network cache data (from the server)
- If a client doesn't find data in its cache it can request another client
  - Since each client has only a subset of data such "forwarding of requests" makes sense only if you know what data is held in another client's cache
  - Instead of exchanging all the data clients can exchange Bloom filters.

Bloom Filters – Applications [3]

4. Peer-to-Peer Systems – Distributed Hash Tables

- In a P2P system, when a query (requesting an object) is issued, first the object must be located
  - Notionally, every peer maintains a hashtable (mapping object IDs to peers)
- As the P2P system gets large maintaining a hashtable in every peer is costly
  - Why?
- An alternative is to maintain a Bloom filter which may result in queries of non-existent objects to remote peers
  - This is acceptable because P2P systems use redundant copies and replicated queries.

LAS VEGAS VS. MONTE CARLO

- Quicksort:
  - Randomization for improved performance – correctness not altered
- Hashtables (for unordered dictionaries) :
    - Any 1-to-1 mapping will yield a table but a good hash function should yield a “uniformly random” distribution
    - Universal hashing chooses hash function “randomly”
  - Both of the above are “performance” enhancements
- Both of the above are examples of Las Vegas techniques.
- Monte Carlo Techniques
  - e.g. Bloom Filter - Randomization yields a probabilistic algorithm i.e. that may not produce correct results always.