CS 493: Algorithms for Massive Data Sets	**Local Models, Bloom Filter**
Feb 21, 2002	**Scribe: Qin Lv**

1 Local Models

In global models, every inverted file entry is compressed with the same model. This work wells when assuming uniform distribution of gaps, i.e. the gap sizes are randomly distributed.

However, the distribution of gaps depends on the listing. If we sort URLs into lexical order, URLs from the the same site are likely to be close to each other. Then the distribution of gaps tends to have this "clustering" property. In order to utilize such property, people use local models, where each inverted file entry uses its own model. Generally, local models outperform global models but are more complex to implement.

Golomb code: works well if most gaps are $\leq b$, but it's not good for large gaps. γ-code works better with large gaps.

Skewed-Bernoulli is a combination of the two.

$$V_T = (b, 2b, 4b, \ldots, 2^i b, \ldots)$$

1.1 Local Hyperbolic Model

This model assumes that the probability of a gap has size i is proportional to $1/i$. i.e.,

$$Pr(i) \sim \frac{1}{i}, for i = 1, 2, \ldots, m$$

Let m be the truncate point of gap size, then

$$Pr(i) = \frac{1}{i \sum_{j=1}^{m} \frac{1}{j}} \approx \frac{1}{i \times log_e^m}$$

What should m be?

$$\begin{aligned} E[\text{gap size}] &= \sum i \times Pr(i) \\ &\approx \sum i \times \frac{1}{i \times log_e^m} \\ \frac{m}{log_e^m} &= \frac{N}{f_t} \end{aligned}$$

where N is the number of documents, and f_t is the frequency of terms.

We can then use observed frequencies to build the code. But one problem with this scheme is that we have to do it at a *term by term* base.

To solve this problem, we can batch terms (with same frequencies) together. An even better method is to use logarithmic batching, i.e., we group terms according to logarithmic intervals $[2^i, 2^{i+1})$. In practice, this method uses only about 20 codes.

All the models we have looked at so far are zero-order model, which means that each symbol is independent, and they do not use context information. We might be able to design better schemes if we can use information of previous gap distribution. Such compression schemes are called *context sensitive* compression schemes.

For the parameterized global model such as Golomb code, we can choose the parameter b to be the average of the previous K gap sizes. But there is a problem. Suppose we have many small gaps followed by a large gap, b could become 1 for the small gaps, which is the same as unary coding, then, when it comes to the large gap, it's going to use a huge code for that large gap. This problem can be alleviated by using Skewed-Bernoulli code.

1.2 Interpolative Code

With this scheme, we assume that we know the complete inverted file entry, and we recursively calculate ranges and code them with minimal number of bits.

e.g. assume we have 20 documents in total, and a given inverted file entry has 7 documents

$$< 7; 3, 8, 9, 11, 12, 13, 17 >$$

then the gaps are

$$< 7; 3, 5, 1, 2, 1, 1, 4 >$$

note that if we know the 4th number is "11" and the 6th number is "13", there is only one possible value ("12") for the 5th number.

We recursively halve the range and calculate the lowest possible value and highest possible value for the number in the middle. So for the example above, we first look at the 4th number "11", then "8", which is the middle number in the left 3 documents, then "3", "9", then "13" (the middle number in the right 3 documents), and finally "12", "17".

We use (x, lo, hi) to denote that x lies in the interval $[lo, hi]$. Then for the example above, we can get the following sequence:

(11,4,17) since 11 is the 4th number among the 7 documents, and there are 20 documents in total.

(8,2,9); (3,1,7); (9,9,10); (13,13,19); (12,12,12)

Interpolative code can achieve the best compression ratio, but it has longer decoding time.

Why the (x, lo, hi) coding is the best coding? Intuitively, for a sorted sequence of numbers within 1 and N, the median is likely to be $N/2$. And the probability $Pr(x \text{ is the median})$ is much higher if x is close to $N/2$. Then, using Huffman coding, we can get very short code.

2 Performance of Index Compression

Since local models use more information (such as the previous distribution of gap), they perform better than global models. Global observed frequency model is only slightly better than γ or δ codes. Both γ and δ codes have performance close to the local schemes, but are outperformed by the local schemes.

And the performances for the various local schemes are:

$$\begin{aligned} Bernoulli &< Hyperbolic \\ &< Skewed - Bernoulli \\ &< batchedfrequencies \\ &< Interpolativecode \end{aligned}$$

3 Hashing Based Signature Schemes

As for the purpose of indexing, a document is just a set of terms, and the database is some representation of such sets so that queries can be answered efficiently. While sometimes we want *exact* answers to our queries, there are many situations when some probability of mismatch can be allowed. We now discuss a scheme that is used for the later case[1].

3.1 Bloom filters

A Bloom filter represents a set S of n elements by a bit vector V of size m, initially all set to 0. A Bloom filter uses k independent hash functions $(h_1, h_2, \ldots, h_k)$ with range $\{1, 2, \ldots, m\}$. We assume these hash functions map each item in the universe to a random number uniform over the range $\{1, 2, \ldots, m\}$.

For each element $e \in S$, the bits $h_i(e)$ are set to 1 for $1 \leq i \leq k$. To check if an element x is in set S, we check whether all the bits $h_i(x)(1 \leq i \leq k)$ are set to 1. If not, element x is definitely NOT in the set S. Otherwise, we assume that x is in S. However, it's possible that x is actually not in the set, but all its k corresponding bits are set (because some other elements are hashed to those bits). So a Bloom filter may yield a *false positive*, where it suggests that an element x is in S even though it is not. For many applications, this is acceptable as long as the probability of a false positive is sufficiently small.

Bloom filters are widely used in many applications such as distributed caches.

Now given the scheme, what is the probability a *false positive*? Since there are n elements and k hash functions, we need to set $n \times k$ bits. Assuming purely random hash functions, each time we set a bit, the probability that a particular bit is not set is $1 - 1/m$. So, after

[1]reference: "Compress Bloom Filter", by Michael Mitzenmacher.

all the elements of S are hashed into the Bloom filter, the probability that a particular bit is still not set is

$$(1 - \frac{1}{m})^{nk} \approx e^{-\frac{nk}{m}}$$

let $p = e^{-\frac{nk}{m}}$, then the probability of a false positive (this is when all the k corresponding bits are set) is

$$\left(1 - \left(1 - \frac{1}{m}\right)^{nk}\right)^k \approx \left(1 - e^{-\frac{nk}{m}}\right)^k = (1-p)^k$$

Let $f = (1-p)^k$. We now use the asymptotic approximations p and f to represent respectively the probability of a bit not being set and the probability of a false positive.

Suppose the number n and m are fixed, how can we choose the number of hash functions k such that the probability of false positive f is minimized?

$$\begin{aligned}
p = e^{-\frac{nk}{m}} &\Rightarrow \frac{1}{p} = e^{\frac{nk}{m}} \\
&\Rightarrow ln\frac{1}{p} = \frac{nk}{m} \\
&\Rightarrow k = -\frac{m}{n} lnp \\
f &= (1-p)^k = (1-p)^{-\frac{m}{n} lnp} \\
&= e^{-ln(1-p) lnp \frac{m}{n}}
\end{aligned}$$

Since m and n are fixed, f is minimized when $ln(1-p)lnp$ is maximized.

$$\begin{aligned}
\frac{d}{dp} ln(1-p)lnp &= \frac{ln(1-p)}{p} - \frac{lnp}{1-p} = 0 \\
&\Rightarrow p = 1/2
\end{aligned}$$

so the optimal (i.e. minimal) f is achieved when $p = 1/2$:

$$\begin{aligned}
1/2 = p = e^{-\frac{nk}{m}} &\Rightarrow ln2 = \frac{nk}{m} \\
&\Rightarrow k = ln2(\frac{m}{n}) \\
&\Rightarrow f = (1-p)^k = (\frac{1}{2})^{ln2(\frac{m}{n})} \sim (0.6185)^{\frac{m}{n}}
\end{aligned}$$

3.2 Compressed Bloom filter

Many applications using Bloom filter may need to pass the Bloom filter as a message, and the transmission size $Z(Z \leq m)$ can become a limiting factor. If every bit has the same probability, the Bloom Filter cannot be compressed $(Z = m)$. Suppose, however, if we can choose k such that p, the probability of a bit not being set is not $1/2$, we may be able to compress the Bloom filter before sending it out, thus reducing the transmission size Z.

The lower bound of Z is $m \times H(p, 1-p)$, where

$$H(p, 1-p) = -plog_2 p - (1-p)log_2(1-p)$$

is the *entropy* of the distribution $\{p, 1-p\}$.

Suppose n and the transmission size Z are fixed, what is the best k that minimizes f?

[to be continued in next class...]