Filtros de Bloom Escaláveis

Albert Linde
a.linde@campus.fct.unl.pt

Orientador: Nuno Preguiça
nuno.preguica@fct.unl.pt

1 Introdução

1.1 Filtros de Bloom

Um filtro de Bloom[1] é uma estrutura de dados probabilística eficiente em espaço que é usado para testar se um elemento é um membro de um conjunto. Falsos positivos são possíveis, mas falsos negativos não, ou seja, uma consulta devolve que não está definitivamente no conjunto ou devolve que esta no conjunto(pode estar errado). Os elementos podem ser adicionados ao conjunto, mas não removidos. Quanto mais elementos são adicionados ao conjunto, maior é a probabilidade de falsos positivos. Um filtro de Bloom deve ser criado com uma dimensão suficiente para que, dado o número de elementos que se espera serem adicionados ao filtro, a probabilidade de erro ser inferior ao valor considerado seguro para uma dada aplicação.

1.2 Filtros de Bloom Escaláveis

Os Filtros de Bloom Escaláveis [2] são uma variante de Filtros de Bloom que conseguem adaptar-se dinamicamente ao número de elementos inseridos, mantendo um limite à probabilidade de erro. A técnica consiste em criar sequencias de filtros de Bloom com capacidades crescentes e menor probabilidade de erro. Assim conseguimos por uma probabilidade de erro máxima inicial independentemente do número de elementos inseridos posteriormente.

2 Implementação

Neste trabalho foram feitas duas implementações de filtros de Bloom escaláveis, uma em Java e outra em C++. Em Java foi feita um teste inicial para descobrir qual estrutura de dados a usar, tendo sido comparada uma aproximação com BitSet e com vectores de inteiros,de long, entre outros, tendo sido que o vector em inteiros deu os melhores resultados. As duas implementações têm uma interface compatível com uma implementação de filtros de Bloom popular disponível na Internet[3]. Ambas têm construtores e métodos de forma a dar ao utilizador das classes um bom controlo sobre a criação e expansão das estruturas. As duas implementações usam como função de hash o SHA-512.

2.1 JSON

Como foi feita uma implementação em duas linguagens e pensada no âmbito de sistemas distribuidos, queremos ter alguma forma de usar um filtro criado numa linguagem na outra, de os guardar em ficheiro para uso posterior ou de os transferir entre dois processos dum sistema distribuído. Assim as duas implementações têm a capacidade de serializar o conteúdo usando uma representação JSON compatível entre as duas e criar um novo filtro de Bloom a partir duma representação serializada.

2.2 Pacotes usados

Na implementação em java usa se o pacote json-simple-1.1.1. Em C++ usou-se a biblioteca jsoncpp-src-0.6.0-rc2 e openssl-0.9.8k.

3 Conclusões

Fizeram-se teste de desempenho para comparar um filtro de bloom com um filtro de bloom escalável e para comparar o tempo de execução em Java com o de C++. Os resultados destes testes podem ser encontrados em anexo

3.1 Continuação do projecto

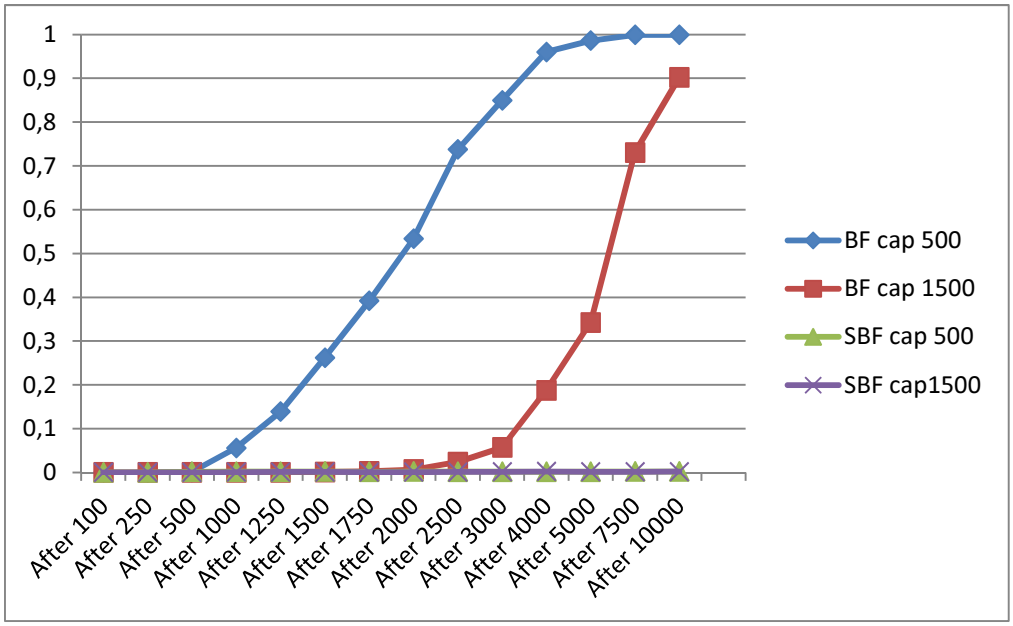
No momento está a ser estudado a introdução dos bloom filter escaláveis em C++ no projecto ChainReaction[4]. Este projecto implementa uma key-value store distribuida que usa Filtros de Bloom para guardar dependências de escrita em objectos. Sabendo se objectos não são dependentes de outros, as escritas podem ser feitas em paralelo. Uma pequena margem de erro diminui a quantidade de vezes que uma escrita tem que preceder a outra erradamente, porque com um erro maior há mais colisões fazendo como que haja mais dependências erradas. Onde os filtros escaláveis podem fazer a diferença é que os filtros usados podem encher facilmente quando o número de dependências cresce, ou seja, a possibilidade de escrita concorrente no servidor diminui porque a probabilidade de erro aumenta nos filtros porque estes não escalam. Já se identificou onde é necessário alterar para permitir a substituição de filtros normais por filtros escaláveis.

4 Referências

[1] B. H. Bloom, Space/time trade-offs in hash coding with allowable errors, Commun. ACM 13 (7) (1970) 422-426.
[2] P. S. Almeida, C. Baquero, N. Preguiça and D. Hutchison, "Scalable Bloom Filters", Inf. Process. Lett., vol. 101, no. 6, pp. 255-261, 2007.
[3] https://github.com/magnuss/java-bloomfilter
[4] S. Almeida, J. Leitão, Luís Rodrigues, ChainReaction: a Casual+ Consistent DataStore based on Chain Replication

ANEXO

Gráfico 1. Probabilidade de erro após inserção de X elementos, com o Filtro de Bloom e o Filtro de Bloom Escalável.



A experiência feita consiste em adicionar um número de elementos e depois verificar a probabilidade de erro da estrutura. Assim, por exemplo, em 'After 2000' temos as probabilidades de erro após serem inseridos 2000 elementos. Os testes foram feitos inicializando as estruturas de dados com probabilidade de erro inicial igual a 0.1% (0.001). BF cap Y é um Bloom Filter[1] inicializado com capacidade esperada de Y elementos, SBF é o ScalableBloomFilter implementado. Neste gráfico conseguimos ver facilmente que quando o número de elementos inseridos é superior ao número esperado a probabilidade de erro aumenta rapidamente, chegando a um ponto em que o filtro dá sempre um resultado errado. Os filtros de bloom escaláveis pelo contrário, o erro ao longo da execução apoxima-se ao erro requerido.

Comparação do tempo de execução de Java e C++.

	Java	C++
Inserir 10000 elementos.	210 ms.	50 ms.
Verificar 10000 elementos.	82 ms.	20 ms.
Traduzir para JSON (com 10.000 elementos).	15 ms.	20 ms.
Traduzir de JSON para objecto (com 10.000 elementos).	2 ms.	20 ms.
Inserir 50000 elementos.	375 ms.	189 ms.
Verificar 50000 elementos.	180 ms.	124 ms.
Traduzir para JSON (com 50.000 elementos).	20 ms.	274 ms.
Traduzir de JSON para objecto (com 50.000 elementos).	10 ms.	231 ms.

Com os resultados podemos ver que o tempo de execução é melhor na versão em C++. A excepção é quando estamos a criar a string json e a devolver um objecto a partir do json, ai a biblioteca em Java usada escala melhor que a em C++.

[1] `http://github.com/magnuss/java-bloomfilter`