Jump Consistent Hash

@_hitsumabushi_

05/Feb/2015

資料

“A Fast, Minimal Memory, Consistent Hash Algorithm” John Lamping, Eric Veach, 2014 http://arxiv.org/abs/1406.2294

このスライドの内容

Consistent Hash Algorithm の一般的な内容と、上記論文の手法の紹介

モチベーション

分散 Key-Value ストアを良い感じに実装したい Key からノードを探す処理をどのように実装すべきか?

Consistent Hashing

以下の mixi のページが端的にまとまっています。
http://alpha.mixi.co.jp/entry/2008/10691/

要点は、

1. 各ノードを $\mathbb{N} mod M$ の値と対応付ける
2. Hash 関数 h で Key k と $\mathbb{N} mod M$ の値を対応付ける。
3. $h(k)$ から一番近いノードに、Value を保存する

というところ。この手法の大事なところは、分散システムとクライアントで同じ Hash 関数を利用することで、自然に Key が保存されているノードがわかることです。また、耐障害性も調整しやすいです。

問題点

実装依存ではあるが、近いサーバーの特定などの問題もあり、log オーダーでの高速なルックアップのためにメモリ上に木を作るものがありました。この場合、メモリ使用料も増えるし、木のリバランスの処理の問題もあります。

Jump Consistent Hash

この問題を解決しようとしたのが本論文で、ハッシュ関数として以下を利用しています。(論文では、この式を導く過程も詳しく書かれている。) よくわからない数字が出てきているところは、擬似乱数を生成する処理として、線形合同法を使おうとしているだけです。

実際のコード

```
int32_t JumpConsistentHash(uint64_t key,  int32_t num_bucke
  int64_t b = -1,  j = 0
  while (j < num_buckets) {
    b = j
    key = key * 2862933555777941757ULL + 1
    j = (b + 1) * (double(1LL << 31) / double((key >> 33) +
  }
  return b
}
```

メリット

1. 高速 はやい。log オーダーで計算できる
2. 省メモリ やすい。キーとバケット数があれば、その場で計算する
3. データを保存しないうまい。キャッシュを汚す可能性が減り、キャッシュヒット率が上がる

制限

1. 見ての通り、ノードは必ず数値で表されている必要がある
2. 1~N を使うとして、途中に空きがあってはいけない
3. 2 と同じことだけど、ノードの削除のロジックは考える必要がある (途中の空きが出てしまうことになるので。)

References

1. “A Fast, Minimal Memory, Consistent Hash Algorithm”, John Lamping, Eric Veach, 2014, http://arxiv.org/abs/1406.2294
2. “スマートな分散で快適キャッシュライフ”, Mixi Engineers Blog, 2008, http://alpha.mixi.co.jp/entry/2008/10691/