Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web

1 目的

人気 web server のように，短時間に猛烈な数の request を処理しなければならないような server (hos spot) を無くしたい/減らしたい，というのがこの研究の大きな目的になっている．hot spot をなくすための手法として，いくつか cache protocol が考えられているが，それらの問題点を挙げ，その問題点を解決するための tool として，Consistent Hashing と Random Trees を提案している．

2 Random Trees

hot spot の問題を解決するための手法として cache を使うものが多くあるが，中でも tree of cache の手法が scalable な手法として挙げられる．ただし，tree の中でも root に近い node への負荷が大きくなってしまう問題点があるため，その点を改善した Random Trees を本論文で提案している．

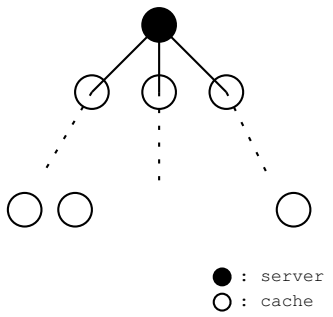

図 1: tree of caches の例

図 1 は tree of caches の構成の例を示している．tree の root を original の page (例えば web page) を持つ server とした tree を cache tree を node として構成している．各 node は受信した request に対する page の cache を持っていない場合は，request を parent の node に forward する．request の forward は page を持つ node に到達するまで forward され，最終的には root までたどり着く．このような動作の tree of caches の問題点は root に近い node ほど負荷が高くな

るという点である．つまり，root に近い node は hot spot になる可能性がでてくる．

Random Trees では page ごとに tree をランダム構成するため，いつも同じ node が tree 内で root の近くにならないようになっている．

protocol　各 page に対し，1 つの root を持つ abstracted tree を構成する．tree は d-ary tree である．各 node と実際の cache の mapping はハッシュ関数 h により行う．各 node は，受信した request 数をカウンタに保存しており，カウンタ値が定数 q を越えない限り，page の replication は行わないとする．これにより，cache に必要なメモリを減らすことができる．browser，cache，server の動作は次のようになる．動作の説明に図 2 を利用する．図 2 はある page に対応する abstract tree で，$d = 3$ である．

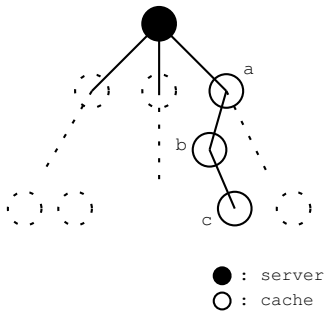

図 2: Random Trees の例

- browser: leaf から root への path をランダムに選び，各 node に対応する cache を h から得る．ここでは，c → b → a と通り，root へ到達する path を選んだとする．次に，request を leaf に対して送信する．request には browser が選んだ path と，各 node に対応する cache の情報が含まれる．
- cache: request に対応する page を持っている場合には，request の送信先へその page を返す．page を持っていない場合には，カウンタの値をインクリメントし，request 内に指定されている path に従って request を次の node へ forward する．request された page を受信することができたら，browser へその page を返す．
- server: request を受信したら，それに対応する page を request の送信元へ返す．

論文内の用語

- cache: オリジナルの page の replica を持つ cache server のこと
- browser: page に対する request を生成するマシン
- server: オリジナルの page を持つマシン

3　Consistent Hashing

従来の hashing based scheme は複数の item を静的な server の集合に一様に mapping するのには優れていたが，server の join/leave がある場合にはすべての item の mapping をしなおさな

ければならないといった問題点があった．そこで，server の join/leave が発生した場合に，item の移動が最小限になるような hashing を考え，これを Consistent Hashing と呼ぶことにする．例えば次のように item と server の mapping を考える．

いま，B と C の item を a と d の server に mapping することを考える．$f(i)$ は i を軸 x 上に配置する関数だとする．図 3 は a，B，C，d それぞれに $f()$ を適用した結果を示している．ここ

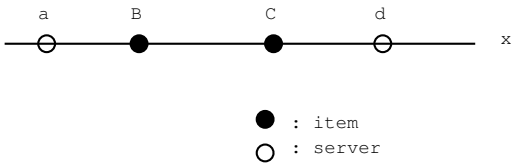

図 3: Consistent Hashing の例

で,「item i は一番近い server s に mapping される」ものとする．すると，item B は server a へ，item C は server d へ mapping される．このように item と server の mapping を決めると，server の join/leave があった際に，すべての item を移動する必要がなく，既存の hash based scheme が持つ問題点を解決することができる．このような hashing を Consistent Hashing と呼ぶ．