社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2006－DSM－40 (8)
2006／3／29

DHTでの部分文字列検索における単語の出現頻度を考慮した負荷分散の一検討

一杉 孝之, 村岡 洋一
早稲田大学 理工学部 情報学科

概要

今や, 低コストのネットワークモデルとしてP2Pが定着している. 特に近年, BitTorrent[1] やeMule[2] など多くのP2Pアプリケーションで, DHT (Distributed Hash Table: 分散ハッシュテーブル) と呼ばれるシステムが実装されるようになってきた. 本研究では, 単語の出現頻度に着目し, これがDHTでの部分文字列検索に与える影響を考察する. そして, この考察でvirtual serversという負荷分散手法の問題点を指摘し, 同時に対策案を提示する. これらを検証するために, Chord[3] というアルゴリズムをベースにしたDHTを形成するJavaプログラムを作成し, これを使用してシミュレーションを行う.

One examination of load-balancing that considers appearance frequency of word in substring search of DHT

Takayuki Hitosugi, Youichi Muraoka
Department of Information and Computer Science, Science and Engineering,
Waseda University.

abstract

Now, P2P is established as a low-cost network model. Especially, the system that is called DHT (Distributed Hash Table) has come to be mounted by a lot of P2P applications such as BitTorrent[1] and eMule[2] in recent years. In this research, we pay attention to the appearance frequency of the word, and consider the influence that it gives to the substring search of DHT. The problem of the load-balancing technique named virtual servers is pointed out by this consideration, and the measures idea is presented. To verify these, we make the Java program that constructs DHT based on the algorithm named Chord[3], and simulate with it.

1 序言

本研究では, DHTでの部分文字列検索における負荷分散に焦点をあてる. ここでの負荷とはインデックスサイズを意味する.

本稿の目的は次の3点である.

1. 部分文字列の出現頻度の偏りに対してvirtual serversという負荷分散手法が耐性を持たないことを検証する
2. 逆に耐性を持ち, 且つ追加のメンテナンスが不要な手法を提案および検証する
3. 上記の偏りがある場合にvirtual serversより本提案手法が優れた負荷分散の効果を与えることを示す

これらを達成するために本研究では, Chordベースの DHTを形成するJavaプログラムを作成し, これを使用してシミュレーションを行う.

2 関連研究

2.1 部分文字列検索

通常の DHT の検索は完全一致のみである. [4] と [5] では, この解決例として, 部分文字列検索の方式を提案している. このどちらの方式でも, あらかじめ部分文字列を抽出してそれらのハッシュ値から複数のアイテム (コンテンツ所持ノードへのポインタもしくはコンテンツそのもの) を登録しておく, という共通動作を行う. それにより, 部分文字列のハッシュ値を探索することで, アイテムを発見できるようになる.

2.2 virtual servers

DHT における負荷分散の代表的な従来研究に virtual servers と呼ばれる手法がある.

DHT では, 各ノードはハッシュ空間の一部分を担当していて, コンテンツの key がその区間内となるアイテムをインデックスに登録する. 以降, この区間を担当領域と呼ぶ. key の分布が一様分布に従うならば, 担当領域が広いノードほど登録するアイテム数が多く, 負荷がかかってしまう. したがって, key の散らばりに偏りがないという前提の下では, 担当領域の広さを均等に近づけることで負荷が均等に分散する. しかし, id を無作為に決めた場合, 担当領域の広さの分布は幾何分布に従う [6]. id を作為的に決めようにも, ノードが自発的に参加および離脱をする環境では, その分布を均一にすることは困難である.

virtual servers では, それぞれのノードが $\Theta(\log N)$ 個の仮想的なノードを持つ. これら仮想ノードは実ノードと同じように振る舞う. つまり, id やルーティングテーブル, インデックスを持つ. さてここで, 仮想ノードについても担当領域の広さの分布は幾何分布に従う. だが, 実ノードとそれが持つ仮想ノード群を合わせて考えると, 分布が相殺されてその広さは均等に近づく [6].

2.3 Two Choices

Power of Two Choice 理論 [7][8] を用いた Two Choices[9] という手法もある. この手法はアイテムの値としてコンテンツ自体を記憶する環境を想定している.

通常なら唯一のハッシュ関数を用いて一意の key を算出するところを, d 個のハッシュ関数で d 個の key を求める. これらの key について担当ノードを探索する. このノード群がアイテムの登録先の候補になる. そして, 各候補の負荷を調べて, 最も負荷が低いノードにアイテムを登録する. 残りの候補には redirection pointer を記憶させる. redirection pointer とはアイテムを登録したノードへのポインタであり, コンテンツを記憶するよりオーバーヘッドが低い.

ノード数を n とすると, ノードの最大負荷は高い確率で高々 $\log(\log n/\log d)+\Theta(1)$ になることが検証されている.

2.4 prefix of key

この小節で紹介する手法 [5] は探索のクエリが届く頻度の均等化を目指している. 便宜上, 本稿ではこの手法を prefix of key と呼ぶことにする.

prefix of key では, ハッシュ空間を均等に分割するようにアイテムの登録先の候補を振り分ける. DHT の探索は, 近くは密に遠くは疎に行われる. そのため, 密の探索が振り分けられることになる. つまり, 探索の大部分である細かいホップを振り分けられるので, ノードからしてみれば探索クエリの届く頻度が分配される. この振り分けをするために, key の prefix を変えるという方法を用いる. 例えば 7bit のハッシュ空間で 3bit の prefix を変える場合には, ある key の残る 4bit の suffix が 0010 とすると, {0000010, 0010010, 0100010, 0110010, 1000010, 1010010, 1100010, 111010} という 8 通りの新しい key が生成される. Chord や Kademlia[10] などのアルゴリズムであれば, この方法でハッシュ空間を均等に分割できる. 変更する prefix を $d$bit とおく. d は 160bit の SHA-1 に対して 3 か 4 が良いとされている.

利点には, この手法を適用することによる追加のメンテナンスがないことがあげられる. 提案元の論文 [5] では, 近傍のノードへの複製配置, 古くなったアイテムの更新, という 2 つのメンテナンスの必要性が指摘されているが, どちらも DHT のレベルで解決できるのでここでは考慮しないことにする.

3 考察

3.1 問題点

virtual servers は, 出現する key の頻度が一様分布に従うことを前提にしている. 本研究が焦点をあてる部分文字列検索では, 例えば自然言語の文章のように, 単語の出現頻度が影響して部分文字列の出現頻度が偏ってしまう恐れがある. 従来の部分文字列検索では, 部分文字列の出現頻度が偏ると key の出現頻度も偏ってしまう. つまり, virtual servers は単語の出現頻度を原因として負荷分散の効果を無くしてしまうかもしれない.

また, Two Choices には候補ノードがハッシュ空間の近傍になり別のハッシュ関数から同一のノードをアイテムの登録先に選んでしまう可能性がある.

3.2 提案手法

本節では, Two Choices と prefix of key を複合した負荷分散手法を提案する.

この手法は, 同一の key を持つアイテムを複数のノードに振り分けることで, 部分文字列の出現頻度の偏りに耐性のある負荷分散を狙う. まず, Two Choices では複数のハッシュ関数を使用してアイテムの登録先の候補を複数用意するが, これを key の prefix を変えるという方法に変更する. すなわち, 唯一のハッシュ関数を使用して 1 つの key を求めた後, その key の $d$bit の prefix を変更すれば 2^d 通りの key を生成できる. 言い換えれば, アイテムの登録先の候補ノードを 2^d 個用意できる. この方法により, key がハッシュ値の空間を均等に分割するので, 候補が近傍になる事態は起こりにくくなる. 次に, 2^d 通りのうち無作為に g 個の prefix を選んで g 個の key を生成する. この key を基にアイテムの登録先の候補となるノードを探索して, それらの負荷を比較する. そして, 最も負荷の低いノードにアイテムを登録する. Two Choices では残りのノードに redirection pointer を記憶させるが, この動作を行わないことにする. こうすれば, 追加のメンテナンスは不要になり, Kademlia のようにノードが頻繁に加入・脱退する環境を想定したアルゴリズムにも適用させることができる. また, アイテムの値として location を登録する環境にも適用させることができる.

4 評価

4.1 はじめに

DHT での部分文字列検索において, 部分文字列の出現頻度に偏りがある場合に, 以下の事項をシミュレーションによって検証する.

1. virtual servers が与える負荷分散の効果が弱まること
2. 本研究の提案手法が与える負荷分散の効果が 1. に比べて弱まりにくいこと
3. 本研究の提案手法が virtual servers より強い負荷分散の効果を与えること

4.2 環境

部分文字列検索は [5] の方式を採用する. この方式はインデックスを 2 つ持つが, 評価に用いるアイテム数は両方のインデックスのアイテム数を足したものとする.

シミュレーションを行うプログラムとして, Chord ベースの DHT を形成する Java プログラムを作成した. このプログラムは一マシン上に擬似ノードを複数生成する. これらノードは DHT のネットワークを繋ぎ, 各自が保有するコンテンツをこのネットワーク上に公開する. id は無作為な 160bit の数値である. ノードが保有するコンテンツの数は均等である. ノードには部分文字列が与えられ, それを一定の分割数だけ連結して, コンテンツ名を作成する (コンテンツ名をある方法で一定の分割数に切断して部分文字列を抽出する, という動作を模する). 部分文字列の出現頻度は次小節のシナリオに従って入力される. それぞれの部分文字列は, 無作為な $0 \sim 2^{31}$ の数値を 36 進数に変換した文字列で構成される. つまり, 数字 0〜9 とアルファベット $A \sim Z$ から成る, ほぼ 2^{31} 通りの文字列である. Chord の successor リストは, ノード数を N とおくとき $\log N$ 個のリンクを保持することにする. よって, レプリカの数も $\log N$ 個になる. この $\log N$ という数値は, Chord の論文で $\Theta(\log N)$ のときの障害耐性が検証されていることに基づいている.

ノード数を 500 個, コンテンツ数を 10000 個, コンテンツ名の分割数を 10 個に設定する. したがっ

て, それぞれのノードが保有するコンテンツ数は20個であり, すべての部分文字列の数は100000個となる.

4.3 シナリオ

シミュレーションでは, 次に示す負荷分散手法を適用したDHTに, 3通りの部分文字列の出現頻度を入力することで, 前述の事項を検証する.

- **"none"**
  nil の負荷分散手法
  (= 負荷分散手法を適用していない状態).
- **"virtual servers"**
  仮想ノード数 $\log N$ の virtual servers.
- **"g-Choices from prefix of key"**
  $\{d = 3, g = d\}$ のときの本研究の提案手法.

文字列の出現頻度にはzipfの法則を用いる. zipfの法則とは「自然言語の文章中で単語の頻度 (出現回数) を1位, 2位, 3位, ... とランキングしていくとき, 順位が n 倍すると頻度は $1/n$ 倍になる」という法則であり, 単語の出現頻度のモデル化によく利用される.

3通りの部分文字列の出現頻度を次に挙げる部分文字列の分布に定める.

I 一様分布に従う.
すべて異なる文字列.

II zipfの法則に従う.
最大頻度の部分文字列は25%のコンテンツ名に出現する.
ただし, 一つのコンテンツ名に同じ部分文字列は出現しない.

III zipfの法則に従う.
最大頻度の部分文字列は50%のコンテンツ名に出現する.
ただし, 一つのコンテンツ名に同じ部分文字列は出現しない.

測定データとして, ノード毎のアイテム数を集計して, その最大値, 最小値, 平均値 (ただし平均値はすべて等しい), 変動係数 (= 標準偏差 ÷ 平均値) を求める. シミュレーションの試行回数は20回で, 全試行結果についてそれぞれの測定データを求めて, その平均を実際に評価に使用する実験データとする.

最大値と最小値の差が開くほどノードの負荷の局所性が強いことを意味し, 逆に差が縮まって双方が平均値に収束するほど局所性が弱いことを意味する. 局所性が強いとは, 一部のノードが極端に高いもしくは低い負荷を負い, 不公平さが生じている状態を指す. また, 変動係数が0から離れるほど大域的に負荷がばらついていることを意味し, 逆に0に近づくほど負荷が均等に近づいていることを意味する. 以上の2つの観点から負荷分散の効果を評価する.

4.4 結果

本小節では, シミュレーションの結果を示す.

それぞれの負荷分散手法の実験データを図と表で表す. 図は, 最大値, 最小値, 平均値の推移を示したグラフである. 縦軸がアイテム数, 横軸が部分文字列の分布となっている. 右側の分布ほど部分文字列の出現頻度に偏りがある. 表は, 第一列に部分文字列の分布, 第二列に変動係数を持つ. 下行の分布ほど部分文字列の出現頻度に偏りがある.

4.4.1 none

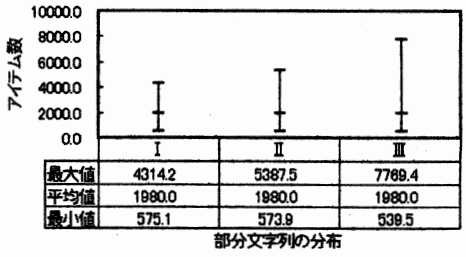

図 1: none

図1と表1はnoneの実験データである.

図1では, 右に移るほど最大値と最小値との差が大きく開いている. つまり, 部分文字列の出現頻度の偏りが大きくなるほど, ノードの負荷の局所性が強くなっている. また, 表1では, 下行に移る

部分文字列の分布	変動係数
I	0.326984
II	0.401022
III	0.551171

表 1: none

ほど変動係数が大きくなっている. すなわち, 部分文字列の出現頻度の偏りが大きくなるほど, ノードの負荷にばらつきが起きている. 以上より, 部分文字列の出現頻度の偏りが原因で, ノードの負荷も偏ってしまうことを確認できた.

4.4.2 virtual servers

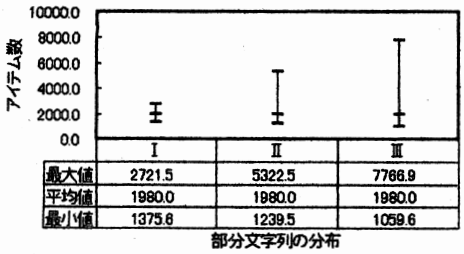

図 2: virtual servers

部分文字列の分布	変動係数
I	0.111579
II	0.245552
III	0.446437

表 2: virtual servers

図 2 と表 2 は virtual servers の実験データである.

図 2 では, 部分文字列の出現頻度に偏りがないときには, 最大値と最小値の差は none と比較すると半分以下になっている. 最大値も最小値も none の場合より平均値に近い値である. 加えて, 表 2 では, 変動係数は none と比較すると 1/3 程度に抑えられている. このことから, virtual servers は, key が一様分布に従うという条件の下では, 優れた負荷分散の効果を与えることがわかる. しかしながら, 文字列の出現頻度に偏りが現れると, 実験データは none に近づいていく. 特に, 最大値は平均値から大きく離れて none とほぼ同じになっている. この原因は, 頻出の部分文字列から生成された頻出の key が同じノードに記憶されるために, その key の数が十分に大きい場合にはその数が最大値になるからである. つまり, virtual servers では同一 key が同一ノードに記憶されるために, 頻出の key が存在した場合にはノードの負荷の局所性が強くなってしまう以上より, DHT での部分文字列検索において部分文字列の出現頻度に偏りがある場合に, virtual servers の負荷分散の性能が弱まる, ということを検証できた.

4.4.3 g-Choices from prefix of key

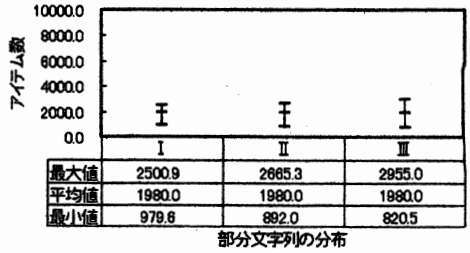

図 3: g-Choices from prefix of key

部分文字列の分布	変動係数
I	0.129280
II	0.159475
III	0.199652

表 3: g-Choices from prefix of key

図 3 と表 3 は g-Choices from prefix of key の実験データである.

この実験データでも prefix of key と同様に, 部分文字列の出現頻度に偏りが現れたときに, 負荷分散の効果は virtual servers ほど弱くはならない. さらに, この g-Choices from prefix of key は強い負荷分散の効果を与えている. 最大値は 2000 台後半に, 変動係数は 0.1 台に維持している. また, 最

小値も prefix of key の場合より平均値に近い．このことから，ノードの負荷比較がアイテムの登録先ノードの振り分けを均一化していること，そしてそれが負荷分散の効果を上げていることがわかる．最大値が 2000 台後半，変動係数が 0.1 台という値は部分文字列に偏りがないときの virtual servers に劣らない性能である．以上より，DHT での部分文字列検索において部分文字列の出現頻度に偏りがある場合に，本研究の提案手法が virtual servers よりも優れた負荷分散の性能を持つ，ということを検証できた．

4.5 まとめ

第 4.4 小節のシミュレーション結果から第 4.1 小節で述べた 3 つの事項をすべて検証できた．

5 結言

本稿では，DHT での部分文字列検索において，単語の出現頻度が与える影響とその対策について述べた．

virtual servers のような，ハッシュ空間を分配する方針の負荷分散手法は，key の分布が一様分布に従うことを期待している．ここで，部分文字列検索を導入したときに，単語の出現頻度が原因で key の分布にも偏りが現れると，このような負荷分散手法は効果を失うことになる．そこで本研究では，同一の key を持つアイテムを複数のノードに振り分けて登録することで，部分文字列の出現頻度の偏りに耐性を持つ手法を提案した．さらにこの提案手法は，アイテムの登録先の候補ノードがハッシュ空間に均等に散らばる点，追加のメンテナンスが不要である点の 2 つの利点を有する．そして，この手法は，部分文字列検索において部分文字列の出現頻度に偏りがある場合に，virtual servers を超える負荷分散の効果を与える．

第 4 節では，以上の考察をシミュレーションによって検証した．

参考文献

[1] BitTorrent, http://www.bittorrent.com/.

[2] eMule, http://www.emule-project.net/.

[3] I. Stoica, R. Morris, D. R. Karger, M. F. Kaashoek, and H. Balakrishnan: “Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications”, In Proc. of *ACM SIGCOMM 2001*, pp. 149-160 (Aug. 2001).

[4] M. Harren, J. M. Hellerstein, R. Huebsch, B. T. Loo, S. Shenker, and I. Stoica: “Complex Queries in DHT-based Peer-to-Peer Networks”, In Proc. of *IPTPS 2002*, pp. 242-259 (Mar. 2002).

[5] A. Nandan, M. G. Parker, G. Pau, and P. Salomoni: “On Index Load Balancing in Scalable P2P Media Distribution”, Special Issue of the Multimedia Tools and Applications Journal on *Advances in Consumer Communications and Networking* (Apr. 2005).

[6] 岡敏生, 森川博之, 青山友紀: “分散ハッシュテーブルの軽量な負荷分散手法の検討”, 電子情報通信学会技術研究報告, IN2003-189 (Feb. 2004).

[7] Y. Azar, A. Z. Broder, A. R. Karlin, and E. Upfal: “Balanced allocations”, In Proc. of *the 26th Annual ACM Symposium on the Theory of Computing*, pp. 593-602 (May 1994).

[8] M. Mitzenmacher, A. W. Richa, and R. Sitaraman: “The Power of Two Random Choices: A Survey of Techniques and Results”, *Handbook of Randomized Computing*, vol. 1, edited by P. Pardalos, S. Rajasekaran, and J. Rolim, Kluwer Academic Pub., pp. 255-312 (June 2001).

[9] J. Byers, J. Considine, and M. Mitzenmacher: “Simple Load Balancing for Distributed Hash Tables”, In Proc. of *IPTPS 2003*, pp. 80-87 (Mar. 2003).

[10] P. Maymounkov and D. Mazieres: “Kademlia: A Peer-to-peer Information System Based on the XOR Metric”, In Proc. of *IPTPS 2002*, pp. 53-65 (Mar. 2002).