Vol.2014-OS-128 No.10
2014/3/7

需要の集中を考慮した 分散 Key-Value Store における レプリカの動的配置手法

白松幸起[1]　堀江光[1]　河野健二[1,2]

概要：DHT には特定のコンテンツに対してリクエストが集中すると，担当ノード及びその周辺 ノードへ負荷が集中するという問題点がある．既存の対策として，あらかじめリクエスト集中時の負荷を予測し，負荷分散のためのレプリカを配置しておく手法がとられているが，突発的なリクエストの集中を引き起こす原因は多岐にわたり，リクエスト集中時の負荷の予測は一般には極めて困難である．本研究では，あらかじめレプリカを配置することなく，リクエストの集中に対して耐性の高いレプリカの配置手法を提案する．提案手法では動的にレプリカを配置することで，オーバープロビジョニングを抑える．同時に，Chord の特性を利用し，リクエスト配送経路となりやすいノードへレプリカを配置する．また，配置したレプリカへのアクセス率を高めるため，新規ノード加入時にノード分布の均等化を行なう．提案手法の負荷削減効果を示すため Overlay Weaver [1] を用いてシミュレーション実験を行った．担当ノードへのリクエスト数の平均において，提案手法では既存手法よりも約 20% 以上負荷を低減することができた．

キーワード：分散ハッシュテーブル，レプリケーション，負荷分散

1. はじめに

近年，個人間の連絡，ショッピング，企業間取引など，日常生活からビジネスまで社会生活の様々な局面でインターネットが欠かせない生活基盤となっている．中でも Web サービスの規模は年々増大する一方であり，Amazon や 楽天などのショッピングサイト，Facebook や mixi，Twitter といったソーシャルネットワーキングサービスでは，日々大量のデータが作り出されている．たとえば，Amazon ではピーク時には数千万件以上のショッピングカートのデータを保持し，1 日の購入件数も 300 万件をゆうに超えている [2].

そのような大規模な Web サービスには負荷分散やスケールアウトのしやすさに優れた分散 Key-Value Store (KVS) が用いられることが多い．Web サービスの急速な大規模化に計算機の性能向上などスケールアップだけで対応することはマシンの性能上限の面でも費用対効果の面でも難しく，このようなサービスでは一般にスケールアウトでの対応が行われる．たとえば Web サーバにおいてリクエストが増えてきた時，サーバの台数を増やし並列に処理することで性能を上げる，または負荷が高い状況でも性能を維持することが可能となる．分散 Key-Value Store としては，たとえば Amazon の Dynamo [2]，Google の Bigtable [3]，Facebook の Cassandra [4] などが利用されている．分散 Key-Value Store とは複数のノードで構成され，指定した Key に対応づけた Value の保存・取得のみを行うことができる単純なデータベースである．従来 Web サービスではデータベースとして Relational Database (RDB) が用いられてきた．Relational Database は複雑な検索や集計に適しており，銀行取引のようなトランザクションを扱ううえで必要となるデータの一貫性を保証するためのロック機能や処理失敗時のロールバック機能を備えている．しかし，複数のデータベースサーバで処理をおこなう分散環境では，一貫性を保つために各ノードで同期を行う必要が生じる．ノードの台数が増えるにつれてこの同期のためのコストが大きくなっていき，サーバの台数に比例した性能を得ることが困難である．分散 Key-Value Store はデータ構造が単純であり，eventual consistency と呼ばれる緩やかな一貫性を採用していることから，並列に処理を行うことが容易であり，高速に読み書きを行うことができる．また，各ノードが同等の機能を持ち，担当する Key の範囲

[1] 慶應義塾大学
Keio University
[2] JST CREST

© 2014 Information Processing Society of Japan

を変更するだけでデータの分割が容易に行えるため，ノードの追加時に負荷の高い処理を必要としないなどの特徴を持つ．

KVS のスケールアウトの容易さを支えているのはその要素技術の一つである分散ハッシュテーブル (DHT) である．Dynamo [2]， Cassandra [4] などの分散 KVS の基盤技術にも DHT が利用されている．DHT とは，Key と Value のペアをデータとして持ち，それを複数のノードで分散管理するための仕組みである．Key とハッシュ関数から得られるハッシュ値で ID 空間上の配置を定め，各ノードはあるアルゴリズムに従って分割されたそれぞれの担当範囲のデータを管理する．Chord [5] ，Kademlia [6] などの代表的な DHT では，ハッシュ値が衝突することの少ないハッシュ関数を用いれば，ノード数を N としてすべてのノードに $O(\log N)$ の経路長で到達可能なことが証明されている．

DHT には特定のコンテンツに対してリクエストが集中すると，担当ノード及びその周辺ノードへ負荷が集中するという問題点がある．DHT はハッシュ関数を使ってノードもデータも同じ ID 空間上に配置することで，すべてのノードからすべてのノードへ平均的にリクエストが発生することを前提として設計がなされている．しかし，Web コンテンツへのリクエストでは Zipf 分布 [7] のように特定のコンテンツに対してリクエストの大半が集中するということが知られている [8]．ノードへの負荷が許容範囲を超えるとリクエストの応答遅延やサービスの停止につながり，可用性を大きく損なうことになる．

一部のコンテンツへのリクエストの集中に対する既存の対策として，あらかじめ負荷分散のためのレプリカを複数配置する手法がある [9]．この方法はレプリカを用意してあるコンテンツへのリクエストの増加に対して有効であるが，リクエストの集中時以外は余分に配置したレプリカがメモリを浪費するなど，コンピュータ資源を圧迫してしまうということになる．また，どのコンテンツに対するリクエストがどの程度増加するかについて事前に予測する必要があるので，予測を超えるアクセスの増加に対しては十分な効果を得ることができず，遅延時間の増加やサーバーのダウンなどを招くおそれがある．しかし，リクエスト集中時の負荷の予測は一般には極めて困難であるといわれている．これは，突発的なリクエストの集中を引き起こす原因が多岐にわたるためである [10]．

本研究では，あらかじめ複数のレプリカを配置することなく，特定のコンテンツへのリクエストの集中に対して耐性の高いレプリカの配置手法を実現することを目的とする．

ここで，リクエストの集中に対して耐性が高いとは，リクエストの集中時に自律的に負荷を他のノードに分散できることをいう．リクエストの集中に対して高い耐性を持つためには，リクエストが集中した際に，リクエストが経由しているノードに対してレプリカを配置することが必要となる．

また，あらかじめレプリカ配置を行わないことで資源の浪費を抑制し，必要とするハードウェアを削減したり，電力などノードの維持管理コストを抑えることができる．このためには，リクエストの集中を検出し，リクエスト数に応じて動的にレプリカを配置することが必要となる．

本研究では，各ノードがコンテンツへのリクエストの集中を検出し，多数のリクエストが経由しているノードに動的にレプリカを配置する手法を提案する．

提案手法では各ノードが担当するすべてのコンテンツへのリクエスト数を監視し，リクエストが集中するとレプリカを配置する．リクエスト集中の検出を行い，動的にレプリカを配置することで，オーバープロビジョニングを防ぎ，また，リクエスト集中に対して必要に応じた負荷分散を行うことができる．

レプリカの配置について，DHT のリクエスト配送経路に関する特性を活かして，レプリカ数に対し効率的な負荷分散を行う．DHT では，リクエストがあるノードに集中した時，リクエスト配送経路も特定のノードに集中しやすいという特性を利用して，負荷を効率的に分散できるようなノードへレプリカの配置を行う．

また，DHT のリクエスト配送経路に関する特性を最大限活かすために，新規ノードの加入時にノード分布の調整を行う．ノード間距離の最大値を抑えることで，レプリカ配置ノードがリクエスト配送経路上に含まれる可能性を高める．

本実験では，特定のコンテンツにリクエストが集中した際に，提案手法が動的にレプリカを配置し，既存手法よりも効果的にリクエストの分散ができることを示すことを目的とする．多数のノードが一つのコンテンツに対して集中してリクエストを行う環境を想定したシミュレーションを行い，担当ノードに集中するアクセスの変化を既存手法と ID 候補数を変えた提案手法の計 4 手法で比較する．提案手法は，60 分間で約 4 個のレプリカを新たに配置し，既存手法に比べ 20% 以上担当ノードへのリクエストを軽減した．

本論文の構成を以下に示す．2 章では分散ハッシュテーブルとその問題点，及び，既存の対策手法について説明し，3 章では本研究の提案手法について説明し，4 章では提案手法を評価する．5 章で本研究の関連研究を紹介し，6 章でまとめを述べる．

2. 背景

2.1 分散ハッシュテーブル

分散ハッシュテーブル (DHT) とは，Key と Value のペアをデータとして持ち，それを複数のノードで分散管理するための仕組みである．Key とハッシュ関数から得られるハッシュ値で ID 空間上の配置を定め，各ノードはあ

© 2014 Information Processing Society of Japan

るアルゴリズムに従って分割されたそれぞれの担当範囲のデータを管理する．データの検索は Key をもとにして行われる．

DHT は，クライアントサーバモデルに比べ，高いスケーラビリティと耐故障性を実現している．従来のネットワークにおいて主流であったクライアントサーバモデルは，特定の役割を集中的に担当するサーバに対して，利用者の操作するクライアントが要求を送信し，サーバが実際の処理を行った上でクライアントに応答を返す形で処理が行われる．クライアントサーバモデルはサーバが公開されており，これに対して多数のクライアントがアクセスするという特徴を持つ．クライアントサーバモデルでは単一のサーバに対してクライアントからの要求が集中するため，クライアントの数が増えていくと，サーバの負荷もそれに比例して増えていく．このようにサーバとクライアントの機能は明確に区別されており，ほとんどの場合，サーバの数はクライアントの数に対して圧倒的に少ない．そのため，クライアント数が膨大になった場合，サーバのコンピュータ資源あるいはネットワーク回線の能力がシステム全体のボトルネックとなってしまう．また，サーバが単一障害点であり，サーバに問題が生じた場合システム全体が停止しまう．DHT は，クライアントやサーバといった明確な区別を行わず，各ノードがサーバとクライアント両方の機能を果たす．また，ユーザは他のユーザに直接アクセスするという特徴を持つ．DHT では検索のためのインデックス情報などを管理するサーバが存在しないため，各ノードが自律分散的に動作し，ユーザ数が膨大になってもシステム全体は変わらず動作する．また，単一障害点が存在しないため，耐故障性も高い．

また，DHT にはノード数が増加した際にも，高速に検索が行えることが求められる．Chord [5] ，Kademlia [6] などの代表的な DHT では，経路表を適切に構築することで 1 回のルーティングで探索する Key までの残り距離を半分以下にする．これらは，ハッシュ値の衝突が少ないハッシュ関数を用いれば，ノード数を N としてすべてのノードに $O(\log N)$ の経路長で到達可能なことが証明されている．

DHT の代表的なルーティングアルゴリズムである Chord [5] の場合，1 次元のリング状論理空間である ID 空間に各ノードをマッピングする．各ノードは，一つ前のノードの ID の次から自身の ID までの範囲のデータを担当する．こうすることで，ノードが故障などで離脱した際もそのノードの担当範囲は次のノードに引き継がれ，残りの範囲では担当ノードが全く変わらないため，ノードの増減に対する耐性や耐障害性は高い．

Chord でのルーティングは Successor List と Finger Table という 2 種類の経路表を用いる．経路表とは，データやノードを探索する際に用いるノードの ID と IP アドレスのペアのリストのことである．経路表の中で，最も探索対象に近いノードへの通信を繰り返すことで，探索対象へ到達する．Successor List と Finger Table は，Chord の耐故障性及び検索効率の向上を実現している．Successor とは ID 空間上で自身の次のノードのことである．これを辿っていくことによって ID 空間内のすべてのノードへの到達性を保証する．しかし，Successor を一つしか持たない場合，その 1 ノードが故障しただけで，その先へのルーティングを行うことができなくなり，一部のノードが管理するデータに到達できないということが生じる．このようなことを防ぐため，Successor に加え，Successor の次のノード，次の次のノードと順番に Successor を経路表に載せる．それによって，故障したノードがあってもそのノードを飛び越えてルーティングを行うことができるので，耐故障性が向上している．このリストを Successor List といい，ノード数を N として $O(\log N)$ 個のリストを持つことで，全体の半分のノードが故障しても目標ノードまで到達できることが保証されている．

Finger Table とは効率的な探索を実現するための経路表である．n bit ID 空間の場合，2^0 先の ID の担当ノードから 2^{n-1} 先の ID の担当ノードまでの n 個のノードのリストになる．Finger Table を利用すると，1 回のルーティングで目標ノードまでの残り距離を必ず半分以下にすることができる．つまり，ノード数を N とした時，$O(\log N)$ 回のルーティングですべてのノードへたどり着くことができる．

2.2 問題点

DHT には特定のコンテンツに対してリクエストが集中すると，担当ノード及びその周辺ノードへ負荷が集中するという問題点がある．DHT はハッシュ関数を使ってノードもデータも同じ ID 空間上に配置することで，すべてのノードからすべてのノードへ平均的にリクエストが発生することを前提として設計がなされている．しかし，Web コンテンツへのリクエストでは Zipf 分布 [7] のように特定のコンテンツに対してリクエストの大半が集中するということが知られている [8]．さらに，DHT では同一コンテンツに対する探索はどのノードから開始しても同じノードへ到達し，また，担当ノードに近いほどそのリクエストを経由する割合が高くなっている．そのため，担当ノード周辺で負荷が高いノードが生じるということになる．ノードへの負荷が許容範囲を超えるとリクエストの応答遅延やサービスの停止につながり，可用性を大きく損なうことになる．

2.3 既存の対策手法

一部のコンテンツへのリクエストの集中に対する既存の対策として，あらかじめ負荷分散のためのレプリカを複数配置する手法がある [9]．この方法はレプリカを用意してあるコンテンツへのリクエストの増加に対して有効である

© 2014 Information Processing Society of Japan

が，リクエストの集中時以外は余分に配置したレプリカがメモリを浪費するなど，コンピュータ資源を圧迫してしまうということになる．また，どのコンテンツに対するリクエストがどの程度増加するかについて事前に予測する必要があるので，予測と異なる増加に対しては十分な効果を得ることができない．リクエスト集中時の予測についても，一般には極めて困難である．これは，突発的なリクエストの集中を引き起こす原因が多岐にわたるためである [10].

3. 提案

3.1 概要

本研究では，各ノードがコンテンツへのリクエストの集中を検出し，多数のリクエストが経由しているノードに，動的にレプリカを配置する手法を提案する．

リクエストの集中の検出について，各ノードは自身の持つそれぞれのコンテンツについて負荷の監視を行う．本研究では，コンテンツに対する負荷の指標として単位時間あたりのリクエスト数を用いるが，これは実環境においてはその環境においてボトルネックとなる他のリソースに置換可能である．周期的に一定時間内のリクエスト数を集計し，一定値を超えた場合にリクエストが集中していると見なす．リクエストの集中を検出した場合，各ノードは動的にレプリカの配置を行う．リクエストの集中に合わせてレプリカを配置することで，レプリカをプロビジョニングする必要がなくなり，メモリなどの資源の消費量を抑えることができる．また，レプリカの配置を動的に行うため，リクエスト集中に対して都度十分な量のレプリカを用意することができる．

レプリカの配置について，DHT のリクエスト配送経路に関する特性を活かして，レプリカ数に対し効率的な負荷分散を行う．DHT のリクエストがあるノードに集中した時リクエスト配送経路も特定のノードに集中しやすいという特性を利用して，レプリカを配置するノードを適切に選ぶことで，少ないレプリカ数で効率的に負荷を分散することができる．

また，DHT のリクエスト配送経路に関する特性を最大限活かすために，新規ノードの加入時にノード分布の調整を行う．DHT においてリクエスト配送経路が特定のノードに集中しやすいという特性が最も現れるのは ID 空間がノードで埋まっている場合である．しかし，実用上 DHT の ID 空間がノードで埋まった状態で使用されることは考えにくい．ハッシュ関数を用いてノードがまばらに配置される場合，DHT のリクエスト配送経路に関する特性を効果的に利用できるノードが存在しない可能性がある．ノード間距離の最大値を抑えることで，リクエスト配送経路に関する特性を効果的に利用できる距離にノードが存在する可能性を高める．

3.2 着眼点

本研究では，多数のノードが同一の Key を探索すると，Chord の性質上リクエストの配送経路として選択されやすいノードが生じる点に着眼した．

Chord では Key を探索する際に，Successor List と Finger Table という 2 種類の経路表を用いる．Successor とは ID 空間上で自身の次のノードのことである．これを辿っていくことによって ID 空間内のすべてのノードへの到達性を保証する．しかし，Successor を一つしか持たない場合，その 1 ノードが故障しただけで，その先へのルーティングを行うことができなくなり，一部のノードが管理するコンテンツに到達できないということが生じる．このようなことを防ぐため，Successor に加え，Successor の次のノード，次の次のノードと順番に Successor を経路表に載せる．それによって，故障したノードがあってもそのノードを飛び越えてルーティングを行うことができるので，耐故障性が向上している．このリストを Successor List といい，ノード数を N として $O(\log N)$ 個のリストを持つことで，全体の半分のノードが故障しても目標ノードまで到達できることが保証されている．Finger Table とは効率的な探索を実現するための経路表である．n bit ID 空間の場合，2^0 先の ID の担当ノードから 2^{n-1} 先の ID の担当ノードまでの n 個のノードのリストになる．Finger Table を利用すると，1 回のルーティングで目標ノードまでの残り距離を必ず半分以下にすることができる．つまり，ノード数を N とした時，$O(\log N)$ 回のルーティングですべてのノードへたどり着くことができる．

実際の探索では経路長を短くするために Finger Table が主に使用される．Finger Table を用いると，探索する Key の担当ノードまでの距離以下で最大の 2^k 先のノードまでルーティングする．Finger Table は 1 回のルーティングで担当ノードまでの距離を半分以下にするので，あるルーティングで 2^k 移動した場合，担当ノードまでの残りの距離は 2^k 以下であり，次のルーティングは 2^{k-1} 以下の Finger Table を用いることになる．つまり，ルーティングを行うごとに 1 回のルーティングで移動する距離は 1/2 以下になり、探索開始時の移動距離は大きく，担当ノードに近づくほど移動距離が小さくなっている．これを多数のノードで同時に行った場合，1 回のルーティングによる移動が小さくなる担当ノード付近ではリクエスト配送経路となるノードの密度が高くなり，また，各ノードがリクエスト配送経路となる頻度も高くなる．

各リクエストが経由するノードの中に，担当ノード以外にも多数のリクエストの配送経路となるノードがあれば，そのノードにレプリカを配置することで効率的に負荷を分散することができる．これによって，特定のノードへの集中的なアクセスも，各レプリカで負荷を効率的に分散させられ，また，新しく配置したレプリカが参照されないと

© 2014 Information Processing Society of Japan

いったことも減らすことができる.

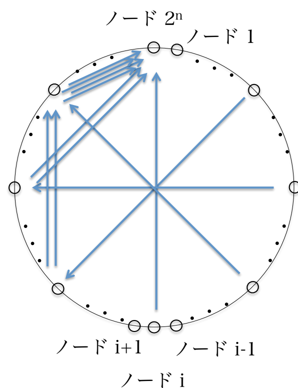

図 1 経路例:n bit ID 空間の場合

ここで, 図 1 に n bit ID 空間においてノード 1 からノード 2^n が順番に並んでいる状況を示す. この時各ノードの Finger Table は更新が十分に行われた後であるとして, 各ノードがノード 2^n にある Key へのリクエストを行った場合の, 各ノードの探索の様子を示す.

この時, 担当ノードに近づくほど経由するリクエストの数は多くなり, ノード i を経由するリクエストの個数は,

$$2^{n-k} \quad (2^{k-1} < 2^n + 1 - i \leq 2^k) \tag{1}$$

となる. これを $n-k$ に関する帰納法で証明する. まず $n-k=0$ のとき, $1 \leq i \leq 2^{n-1}$ であり, $i \leq 2^{n-1}$ のノードは 1 回目のルーティングで 2^{n-1} 先への Finger Table を使用する. よって, $1 \leq i \leq 2^{n-1}$ のノードを他のノードからのリクエストが経由することはなく, 経由するリクエスト数は $2^0 = 1$ となり, 1 式が成立する. 次に $n-k=0 \sim j$ のとき成立すると仮定して, $n-k=j+1$ のときを証明する. $2^{j-1} \leq i \leq 2^j$ のノードが経由するリクエストは, ID が $2^h (0 \leq h \leq j)$ 前のノード の経由するリクエストと自身のリクエストの和である. これは, $\sum_{h=0}^{j} 2^h + 1 = 2^{j+1}$ となり, 1 式が成立する. 以上より, 1 式が証明された.

これは n bit ID 空間において各ノードが同一ノードへのリクエストを行った場合, 担当ノードの Predecessor は全リクエストの半数が経由していき, 担当ノードからの距離が 2^i を超えるごとに経由回数は 1/2 となっていることを示す. このことから, 本研究では効率的なレプリカの配置先として Predecessor を利用する.

3.3 アプローチ

3.3.1 リクエスト配送経路上へのレプリカ配置

レプリカの配置順序について, 本手法では 3.2 章で述べた特性を利用し, リクエスト配送経路となりやすい Predecessor から順にレプリカを配置する. n bit ID 空間においてノード 1 からノード 2^n が順番に並んでいる状況を想定し, 1 つ目のレプリカを Predecessor に配置した後の, 2 つ目のレプリカの配置先について考える. 1 式より, Predecessor の次にリクエスト配送経路となりやすいのは担当ノードの 2 つ前のノード及び 3 つ前のノードであり, リクエストの経由数は Predecessor の半数である. このようにリクエスト配送経路となる頻度は, 担当ノードに近いほど高く, 離れるごとに段階的に 1/2 となっていくので, 新たに配置するレプリカが多くのリクエストを分担するように Predecessor から順に手前にレプリカを配置していく. また, こうすることで, 最も手前のレプリカは他のレプリカの影響を受けることなく, 自身がリクエスト配送経路となるリクエストをすべて処理することになる . 同時に, 担当ノードからは最も遠いので, リクエスト配送経路となる頻度は最も低くなる. そのため, 負荷分散を十分に行えていれば, 最も手前のレプリカにリクエストは集まらなくなり, 過剰なレプリカの配置を抑制することができる.

3.3.2 ノード分布の均等化

リクエスト配送経路上へ配置したレプリカへのアクセス率を高めるため, 本手法では新規ノード加入時にノード分布の均等化を行う.

3.3.1 章で述べたように Chord の特性として Predecessor がレプリカ配送経路となりやすいが, 3.3.1 章の例のように ID 空間がすべてノードで埋まっているような理想的な状態は実用的であるとはいえない. 理想的な状態から離れれば, 担当ノードと Predecessor との距離が長くなり, レプリカを配置しても Predecessor へアクセスが集まらないということが起こりうる. これは DHT ではノードやコンテンツの ID となるハッシュ値の衝突が発生すると管理コストが増えるため, 通常はノード数やコンテンツ数に対して十分に広大な ID 空間中にノードがまばらに存在する状態で使用されるためである.

そこで ID 空間中にまばらにノードが存在する状態でも理想的な状態に近い動作を行うため, 新規ノード加入時には通常 1 つ与えられる ID を ID 候補として複数与える. 新規ノードは ID 候補に隣接するノード間の距離を調べ, 隣接ノードが最も離れている ID 候補を ID として DHT に加入する. これによって, DHT 中の隣接ノード間距離の最大値を低く抑え, Predecessor へのアクセスを集めやすくする.

4. 実験

特定のコンテンツにリクエストが集中した際に, 提案手法が既存手法に比べ効率的に負荷を分散できることを示すため, 実験を行った. 本実験では多数のノードが特定のコンテンツに対して集中的にデータの取得を行うシミュレー

© 2014 Information Processing Society of Japan

ションを行った．シミュレーションには Overlay Weaver を用い，1 台の実機上で 100 台のノードをシミュレートした．

レプリカ配置のスレッショルドは各レプリカへのアクセス数が 15 回 / 分を超えた場合需要の集中とみなし，新たにレプリカを配置した．各レプリカへのリクエスト数は 1 分ごとに集計し，直近 5 分間のデータの平均を用いた．

ワークロードは以下のように作成したものを用いた．

- 100 台のノードを起動し，DHT のネットワークに参加させる．
- ランダムな文字列を Key とし，Key-Value ペアを一組 DHT 上に保存する．
- 1 秒ごとにノードをランダムに選択し，そのノードから先ほどの Key に　対してリクエストを行う

本実験では既存手法と提案手法の候補 ID 数を 3 個 , 5 個 , 7 個用意する場合の計 4 手法に対して 10 種類の Key を用いて各 1 回ずつ計 10 回の実験を行い，その平均値を求めた．実験の評価は担当ノードが 1 分間に処理しているリクエスト数と配置したレプリカ数を用いて行った．

実験結果を図 2 に示す．グラフの縦軸は 1 分間ごとの

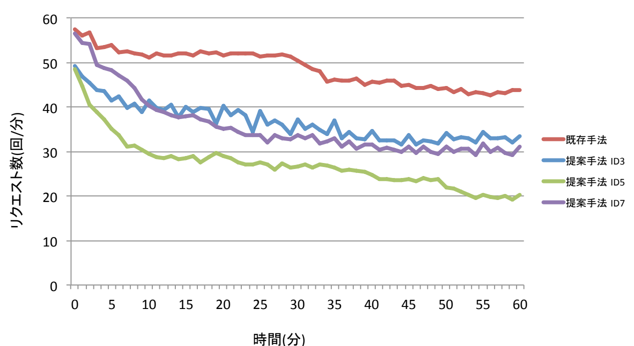

図 2　各手法の担当ノードへのリクエスト数の比較

担当ノードへのリクエスト数の平均を表し，横軸は経過した時間を表す．グラフから 60 分後の担当ノードへのリクエスト数の平均において，ID 候補数 3 個，7 個の提案手法では既存手法よりも約 20%，ID 候補数 5 個の提案手法では既存手法よりも約 50% 負荷を低減することができた．

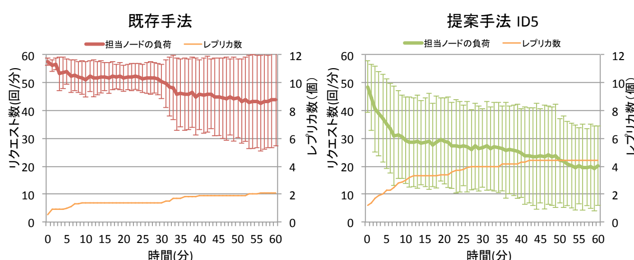

図 3　既存手法と提案手法の比較

既存手法と提案手法の比較結果を図 3 に示す．グラフの縦軸は 1 分間ごとの担当ノードへのリクエスト数の平均を表し，横軸は経過した時間を表す．誤差付きの折れ線グラフは担当ノードへのリクエスト数の平均とその標準誤差，他方の折れ線は配置したレプリカの数の平均を示している．グラフから 60 分後の担当ノードへのリクエスト数の平均において，ID 候補数 5 個の提案手法では既存手法よりも約 50% 負荷を低減することができており，95% 信頼区間も既存手法で 27-60 となっているのに対して，ID 候補数 5 個の提案手法では 6-34 となった．これは既存手法ではレプリカ数を 2 個程度配置した時点で次のレプリカ配置が止まってしまっていたのが，ID5 の提案手法ではレプリカ数を約 4 個まで増やすことができているためであると考えられる．

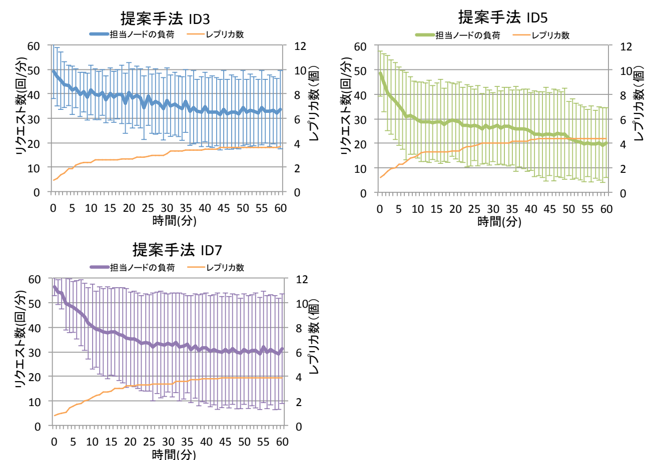

図 4　候補 ID 数の異なる提案手法の比較

候補 ID 数の異なる提案手法の比較結果を図 4 に示す．グラフの縦軸は 1 分間ごとの担当ノードへのリクエスト数の平均を表し，横軸は経過した時間を表す．誤差付きの折れ線グラフは担当ノードへのリクエスト数の平均とその標準誤差，他方の折れ線は配置したレプリカの数の平均を示している．グラフから 60 分後の担当ノードへのリクエスト数の平均において，ID 候補数 5 の提案手法では，ID 候補数 3 及び ID 候補数 7 の提案手法より約 30% 負荷が低くなっている．また，配置したレプリカ数は ID 候補数によらず約 4 個とほぼ同じである．ここで，原理的には候補 ID 数が多いほうがノード分布がより均等に近づき，担当ノードへのリクエスト数の平均についても低く抑えられると考えられるが，図 4 のグラフでは ID 候補数 7 の提案手法よりも ID 候補数 5 の提案手法のほうがリクエスト数は低くなっている．このため，候補 ID 数の増加による負荷分散効率の向上ははやい段階で頭打ちになっており，それ以上は環境によって最適な ID 候補数が変化していると考えられる．

そこで ID 候補数を変化させた場合の隣接ノード間距離の分布を図 5 に示す．グラフより，今回の実験環境では最長ノード間距離が短く，かつ，第 1 四分位点から第 3 四分

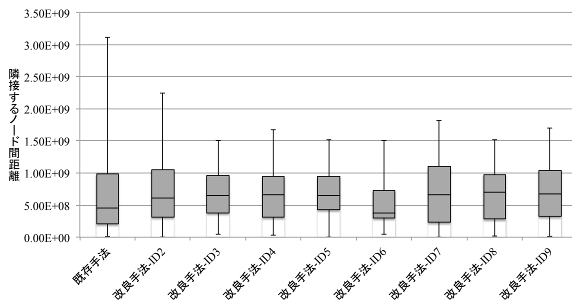

図 5 候補 ID 数によるノード分布の変化

位点がノード間平均距離に最も近くなっていた候補 ID 数 5 の提案手法が最も優れていたといえる．しかし，これは加入するノード数，使用するハッシュ関数，もともとの ID の与え方などの条件により変化するものであり最適な候補 ID 数の選択は今後の課題である．

5. 関連研究

DHT における既存の負荷分散手法として，特定ノードへのアクセス集中に対する負荷分散手法 [11] [12] がある．Effective Load Balancing in P2P Systems [11] では，ファイルのアクセス履歴情報を用いて負荷分散を行うロードバランス機構 Historical information based Load Balancing (HLB) を提案している．これはアクセスの局所性を利用し，今後のアクセスを予測することで負荷分散を効率的に行う．Resilient and efficient load balancing in distributed hash tables [12] では仮想サーバ (VS) の移送による負荷分散手法 group multicast strategy (GMS) を提案している．これは既存の VS の移送における 2 つの代表的な戦略の効率性と耐攻撃性を両立させる手法である．しかし，これらの手法は特定ノードに対する負荷分散手法であり，特定コンテンツに対する負荷の集中を考慮していない．

また，コンテンツの人気度を考慮した負荷分散手法として CoralCDN [9] が用いられている．これは Content Delivery Network (CDN) を形成する各プロキシサーバに Peer-to-Peer (P2P) 技術を応用し，コンテンツ提供者のサーバ (オリジンサーバ) へのリクエスト数を減少させる．しかし，この手法では用意したプロキシサーバの数によって性能が頭打ちになってしまう．

6. まとめ

本研究では，あらかじめレプリカを配置することなく，リクエストの集中に対して耐性の高いレプリカの配置手法を提案した．提案手法ではリクエスト数を監視しレプリカを配置することで，リクエストの集中に動的に対応し，オーバープロビジョニングも抑えた．同時にレプリカの配置先の選択については Chord の特性を利用し，リクエスト配送経路となりやすいノードへレプリカの配置を行なった．また，配置したレプリカへのアクセス率を高めるため，新規ノード加入時にノード分布の均等化を行なった．シミュレーションによる実験で担当ノードへのリクエスト数の平均において，提案手法は既存手法よりも約 20% 以上負荷を低減することができた．

参考文献

[1] Shudo, K., Tanaka, Y. and Sekiguchi, S.: Overlay Weaver: An Overlay Construction Toolkit, *Symposium on Advanced Computing Systems and Infrastructures(SACSIS 2006)*, pp. 183–191 (2006).

[2] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: Amazons Highly Available Key-value Store, *Proceedings of 21st ACM Symposium on Operating Systems Principles(SOSP '07)*, pp. 205–220 (2007).

[3] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A. and Gruber, R. E.: Bigtable: A Distributed Storage System for Structured Data, *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI '06)*, pp. 205–218 (2006).

[4] Lakshman, A. and Malik, P.: Cassandra: A Decentralized Structured Storage System, *ACM SIGOPS Operating Systems Review*, pp. 35–40 (2010).

[5] Stoica, I., Morris, R., Karger, D., Kaashoek, M. F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *Proceedings of the ACM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '01)*, pp. 149–160 (2001).

[6] P.Maymounkov and D.Mazieres: Kademlia: A Peer-topeer Information Systems Based on the XOR Metric, *Proc. IPTPS 2002*, pp. 53–65 (2002).

[7] Zipf, G. K.: *Human Behavior and the Principle of Least Effort*, Addison-Wesley (1949).

[8] Breslau, L., Cao, P., Fan, L., Phillips, G. and Shenker, S.: Web Caching and Zipf- like Distributions: Evidence and Implications, *The Conference on Computer Communications, Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies(IEEE INFOCOM '99)*, pp. 126–134 (1999).

[9] Michael J. Freedman, Eric Freudenthal, D. M.: Democratizing content publication with Coral, *Proc. 1st USENIX/ACM Symposium on Networked Systems Design and Implementation(NSDI '04)*, pp. 239–252 (2004).

[10] Jung, J., Krishnamurthy, B. and Rabinovich, M.: Flash Crowds and Denial of Service Attacks: Characterization and Implications for CDNs and Web Sites, *11th International WWW Conference*, pp. 293–304 (2002).

[11] Damiano Carra, Moritz Steiner, P. M.: Effective Load Balancing in P2P Systems, *IEEE International Symposium on Cluster Computing and the Grid*, pp. 81–88 (2006).

[12] Di Wu, Ye Tian, K.-W. N.: Resilient and efficient load balancing in distributed hash tables, *Journal of Network and Computer Applications*, Vol. 32, pp. 45–60 (2009).

© 2014 Information Processing Society of Japan