構造型P2Pネットワークにおける負荷分散を考慮したXMLデータ処理

呉　俊輝† 天笠　俊之†,†† 北川　博之†,††

† 筑波大学大学院システム情報工学研究科
†† 筑波大学計算科学研究センター
〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †chunhuiwu@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

あらまし 近年，P2P 技術を利用した情報コンテンツの流通に関する研究や応用が盛んに行われている．XML はコンテンツやデータの記述に標準的に用いられてようになっており，P2P 環境における XML データ管理は重要な問題である．我々はこれまで分散ハッシュ表における XML データの効率的な格納・検索手法を提案してきた．しかし，ピア毎に格納されるデータ量に偏りがあり，大規模なデータでは問題となる．そこで，本研究ではピア間の負荷分散を図るため，拡張可能ハッシュ法を利用した XML データの処理手法を提案する．具体的には，ピアに格納されたデータを，拡張可能ハッシュ法に基づき他ピアに再配分する．これによって，各ピアが保持するデータ量を均衡化する．本論文では，提案手法の詳細を述べ，有効性と性能を実験により検証する．

キーワード 構造型 P2P ネットワーク, 分散ハッシュ表, XML, 拡張可能ハッシュ法

Storage and Retrieval of XML Data in Structured P2P Network with Load Balancing

Chunhui WU†, Toshiyuki AMAGASA†,††, and Hiroyuki KITAGAWA†,††

† Graduate School of Systems and Information Engineering, University of Tsukuba
†† Center for Computational Sciences, University of Tsukuba
1–1–1 Tennohdai, Tsukuba, Ibaraki 305–8573 Japan
E-mail: †chunhuiwu@kde.cs.tsukuba.ac.jp, ††{amagasa,kitagawa}@cs.tsukuba.ac.jp

Abstract Recently, there have been extensive work on information management in P2P environments. In such systems, XML (Extensible Markup Language) plays an important role for data description and exchange, and XML data management in P2P networks is an important issue, consequently. For this reason, we have proposed a method for storage and retrieval of XML data using Distributed Hash Table (DHT). However, it has a drawback that storage-load distribution is skewed, because it uses element names as the hash keys. To cope with this problem, in this paper, we propose a scheme for storage-load balancing based on extendible hashing. Specifically, for each peer, we redistribute overflowed data to other peers by extendible hashing. We also show the feasibility of the proposed method by experimental evaluations.

Key words Structured P2P Network, DHT, XML, Extendible Hashing

1. はじめに

近年，計算機性能の向上とネットワークインフラの発展により，大規模なデータに対して高速かつ頑健な処理が可能になる分散コンピューティング技術が注目を集めている．なかでも，Peer-to-Peer(P2P) 技術は，インターネットを通じて大規模な分散コンピューティング環境が容易に構築できる技術として，近年活発に研究されている．

P2P システムは，従来のサーバ・クライアント型システムとの対比で，完全な分散システムであるピュア型と，両者の中間的な性質を持つハイブリッド型とに分類することができる．ハイブリッド型 P2P システムでは，データの所在を管理する仲介サーバの存在を仮定しており，全ての検索処理は仲介サーバを通じて行われ，その結果を利用してピア同士が通信を行う．このため，システムはシンプルであるが，仲介サーバの故障やネットワークにおける局所的な故障がサービス全体の停止の要

因となることがある．一方，ピュア型 P2P システムでは，ピア同士がサーバを介さずに動作するため，局所的な障害があっても全体サービスは持続するという特徴を持つ．

ピュア型 P2P システムは，非構造型と構造型に分類することができる．非構造型では，ピアがフラッディングと呼ばれる方式でメッセージを他ピアへ伝播させる．このため，ネットワークに不要なトラフィックが生じ，帯域が消費されてしまうことが知られている．この問題を解決するため，構造型 P2P と呼ばれる技術が提案されている．特に，**分散ハッシュ表**(Distributed Hash Table; DHT) はその一つとして，効率的なオブジェクト配置と検索の枠組みを提供している．

一方，XML [1] は，1998 年に W3C 勧告となってから現在までの間に急速に普及し，計算機間のデータ交換の標準フォーマットとなっている．XML は，タグを用いて文書やデータの意味・構造を記述するマークアップ言語を記述するためのメタ言語であり，木構造やグラフ構造を記述可能なことから，様々な分野で広く応用されている．最近では，P2P 技術の発展と XML の普及に伴い，こういった XML データを P2P 環境で利用することも一般的に行われるようになっている．例えば，いくつかの科学分野では，実験や観測データを XML で記述して，P2P ネットワークを通じて交換するという応用が出始めている．このことから，今後は P2P 環境で大量の XML データの効率良い格納・検索が重要になってくるものと思われる．

これに対して，我々は P2P 環境における効率的な XML データの格納・検索手法を提案し，プロトタイプシステムの実装や実験によりその有効性を検証してきた [2] [3]．基本的には，XML データからテキスト情報を保持するノードと葉ノードを抽出する．それらを，要素名をハッシュキー，その要素に至るパス式およびテキスト情報の組合せを値として DHT に格納する．しかしながら，この手法では，XML データを要素名をハッシュキーとして DHT に格納するため，同じ要素名のデータがすべて同じピアに格納される．このため，大規模なデータにおいては，データが少数のピアに偏ってしまう欠点を持つ．

そこで本研究では，上記手法にストレージ負荷分散の仕組みを取り込むため，以下の方式を提案する．基本的には，拡張可能ハッシュ法に基づくものである．我々の XML データ処理手法では，ある XML ノードをタグ名によって DHT 内のピアに配置する．あるピアが保持するデータを他ピアに分散するため，格納されている各ノードに関して，その絶対パス式から求められるハッシュ値を用いて，拡張可能ハッシュの手法を適用する．

本論文の構成は次のとおりである．まず第 2 章で DHT や XML の関連技術について基本的事項を説明する．次に第 3 章でこれまでの手法概要を述べ，第 4 章では負荷分散方式を説明する．第 5 章で実験について述べ，第 6 章で関連研究を紹介し，最後に第 7 章でまとめと今後の課題を挙げる．

2. 基本的事項

2.1 分散ハッシュ表

分散ハッシュ表 (Distributed Hash Table; DHT) は，構造型 P2P ネットワークにおけるオブジェクト配置・検索手法である．

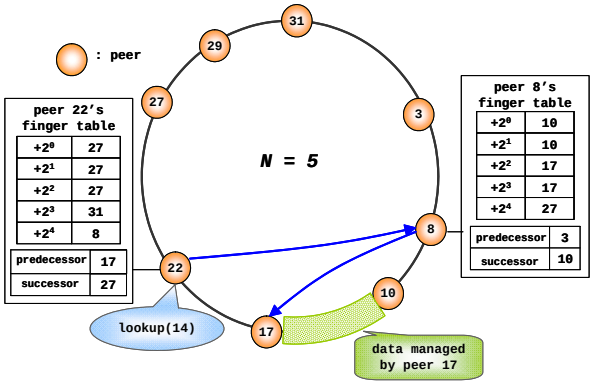


図 1 Chord の ID サークル
Fig. 1 An ID Circle of Chord

DHT では，P2P に参加するピアと検索対象となるオブジェクトを，仮想的なネットワーク（オーバレイネットワーク）上に配置する．具体的には，各オブジェクト（ピア）のオーバレイネットワーク上の位置は，オブジェクト識別子（ファイル名やピアの IP アドレス等）にハッシュ関数を適用して得られるハッシュ値に基づいて求められる．検索時には，各ピアがルーティング表を利用し，検索したいオブジェクトを管理するピアへメッセージを転送することにより検索を行う．DHT には，オーバレイネットワークやルーティング表の構成法にバリエーションがあり，これまでいくつかの方法が提案されている．代表的なものとして，Chord [4]，CAN [5]，Tapestry [6] などがある．以下では本研究で取り上げる Chord について紹介する．

Chord では，オーバレイネットワークとして，2^N 個のオブジェクトからなる仮想的な円構造のハッシュ空間（ID サークル）を用いる．ここで N はスケールファクタと呼ばれる．オブジェクトはハッシュ関数で，ID サークル上に位置する．ID が i のピアは，担当するデータの ID，フィンガーテーブル，前後に位置するピアの ID 情報 ($predecessor$, $successor$) を保持している．またピア i が担当するデータは，前ピア ($predecessor$) から i までのハッシュ値を持つオブジェクトである．フィンガーテーブルには，N 個の他ピアへのリンクがあり，リンク先は $i+2^k$ $(k = 0, 1, 2, ..., N-1)$ 位置を担当するピアである．このため，オブジェクトの検索に要するホップ数は $O(\log_2 N)$ である．

図 1 に Chord の構造とオブジェクト検索の例を示す．ピア 22 がキー 14 の値を得たいとする．ピア 22 のフィンガーテーブル内で最もキー 14 に近いのはピア 8 であるため，まずピア 8 にアクセスする．これを繰り返し行うことで，キー 14 の情報を保持するピアへアクセスする．この例では，キー 14 を保持しているのがピア 17 であるため，最終的にピア 22 はピア 17 にアクセスし，キー 14 に対応する値を取得する．

2.2 XML データと構造概要

XML データは $T = (V, E, r)$ で表される木構造にモデル化される．ここで V はノード集合，E は枝集合，r は XML 木の根ノードであり，r は V の要素である．図 2 に示している，根ノードの子要素には `book`，`music` ノードがあり，シングルクォート内にはテキストノードの値を表している．

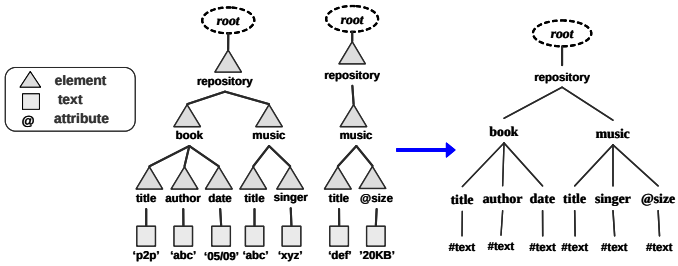


図 2　XML データと Strong DataGuide
Fig. 2　XML data and Strong DataGuide

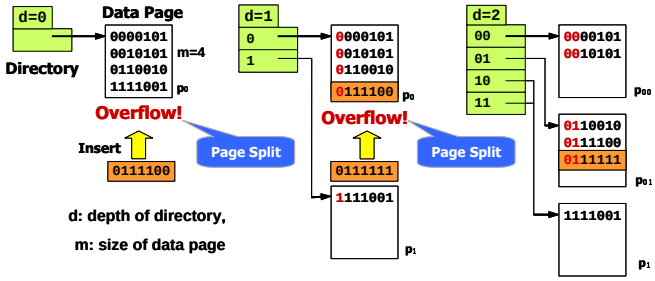


図 3　拡張可能ハッシュ法
Fig. 3　Extendible Hashing

XML データ検索の効率化を図るために，XML データための索引構造がよく研究されている．特に，データの概要を把握するためのいわゆる**構造概要**(Structural Summary) は，これまでに多くの手法が提案されている．本研究では，その中で最も単純なものの一つである **Strong DataGuide** を利用する．これは，概念的には XML 木の根ノードから葉ノードへ向かって，共通のタグを集約することで得られた木構造である．定義等の詳細については文献 [7] を参照されたい．図 2 に XML データとその Strong DataGuide を示す．以下では，Strong DataGuide を DG と呼ぶ．

2.3　拡張可能ハッシュ法

拡張可能ハッシュ法 (Extendible Hashing) [8] は，外部記憶探索に使われるデータの配置手法の一つである．ディレクトリとデータページと呼ばれる 2 階層のデータ構造と，キーを引数とするハッシュ関数からなる．ディレクトリはデータページへのポインタを持つ配列で，キー数に応じて配列の大きさが動的に変化する．データページの領域にはキーに対応する値が格納される．以下は拡張可能ハッシュの定義を示す．

定義　d 次の拡張可能ハッシュ法
キーをもつ項目を M 個まで含むデータページへの参照を 2^d 個もつディレクトリである．各データページの中の項目は，はじめの k ビットが同じ値を持つ．□

キーの挿入は以下のように行われる．まずハッシュ関数によって一様に分布する固定長のビット列であるキーを生成し，上位数ビットを取り出す．ビット列から取り出すビット数 d は，ディレクトリの深さとも呼ばれる．この取り出したビット列をディレクトリ配列の引数として使い，その配列のもつポインタが指すデータページにキーを挿入する．検索および削除も，同様の方法でデータページを特定する．

キー数が増加するとデータページがオーバーフローするが，ディレクトリの大きさを変化させ，データページを分割することで対処する．また反対に，キー数が減少し空き領域ができると，オーバーフローで分割したデータページを結合する．

図 3 に拡張可能ハッシュ法の例を示す．ディレクトリの深さを $d = 0$ から，データページの領域サイズを $m = 4$ とする．データページはすでに図に示したようなキーが格納されているとする．このとき，キー “0111100” を既存のデータページ p_0 挿入すると，オーバーフローが発生し，p_0 の分割が行われる．p_0 に格納されているキーの先頭 1 ビットを見て，先頭 1 ビットが 0 となっているキーは p_0 に配置され，1 となっているキーが p_1 に配置される．この例では，ディレクトリから p_1 をアクセスするための新たなポインタが必要なので，ディレクトリを 2 倍に増やし，ディレクトリのから p_1 へのポインタを張る．続くキー “0111111” の挿入も同様に行う．

3.　DHT による XML データの格納・検索

ここで我々が [2] で提案した，分散ハッシュ表を用いた XML データの格納・検索手法の概要を説明する．基本的には，XML データの葉ノード（テキストノード）単位に分解し，要素名をキー，テキストを含む関連情報を値として DHT に格納する．なお，中間ノードをルートとする部分 XML データの再構成するために，P2P ネットワーク上のピアが保持している全 XML データから DG を構築する．DG は，別途専用の DHT に格納され，データの追加や削除があったときに，随時更新される．

また，各ピアが保持している XML データには，グローバルなスキーマを前提とせず，任意の整形式 XML データを扱うことができるものとする．検索には XPath のうち *child* 軸 (/)，*descendant* 軸 (//) および述語 ([]) を含む XP{/, //, []} [9] を用いる．

3.1　C-DHT による XML データの格納

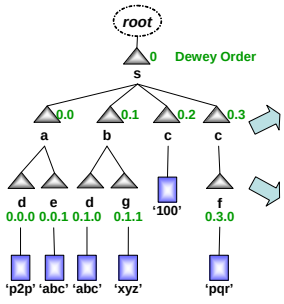
XML データをノード単位に分割した上で，DHT に格納する．このための DHT を **C-DHT**(*Content*-DHT) と呼ぶ．具体的には，XML データのノードのうち，

- テキストノードだけを含む要素ノード
- 属性ノード
- テキストノードを含まない葉要素ノード

などを DHT に格納する．このとき，**要素名**をハッシュキー，

（ピア ID，文書 ID，パス式，順序ラベル，テキスト）

の組を値とする．要素名をハッシュキーとすることで，要素名を元に，その内容を保持しているピア ID とそのピア内での当該文書 ID，その文書内のパス式とノードの位置情報（順序ラベル），テキスト値などを得ることができる．順序ラベルには木構造を表現するための Dewey Order [10] を利用する．特に，パス式と順序ラベルは重要で，これによって後述する述語含むような検索を処理することが可能となる．図 4 に XML データを C-DHT に格納した例を示す．例えば，d ノードはテキスト値 “p2p”, “abc” を持ち，それぞれのパス式や順序ラベルを取



C-DHT

Element	Peer_id	Doc_id	Path	Label	Text
d	1	Doc1	/s/a/d	0.0.0	p2p
e	1	Doc1	/s/a/e	0.0.1	abc
d	1	Doc1	/s/b/d	0.1.0	abc
g	1	Doc1	/s/b/g	0.1.1	xyz
c	1	Doc1	/s/c	0.2	100
f	1	Doc1	/s/c/f	0.3.0	pqr

S-DHT

Element	Path	(Child, Count)
s	/s	(a, 1) (b, 1) (c, 2)
a	/s/a	(d, 1) (e, 1)
b	/s/b	(d, 1) (g, 1)
c	/s/c	(#text, 1) (f, 1)
d	/s/a/d	(#text, 1)
e	/s/a/e	(#text, 1)
d	/s/b/d	(#text, 1)
g	/s/b/g	(#text, 1)
f	/s/c/f	(#text, 1)

図 4　XML データから C-DHT，S-DHT を生成する
Fig. 4　Storing XML in C-DHT and S-DHT

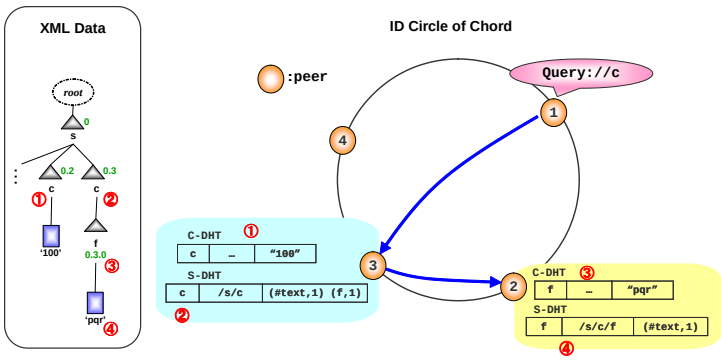


図 5　検索式 q=//c の処理過程
Fig. 5　Query Processing of q=//c

得することができる．

3.2　S-DHT による構造概要の格納

S-DHT(*Structure*-DHT) は，P2P ネットワーク上のピアが保持している全 XML データから構築される *DG* を格納するための DHT である．具体的には，*DG* の各**要素名**をハッシュキー，

（パス式，子ノード集合）

の組を値として格納する．子ノード集合には，子ノードとして現れるノードを列挙するのに加えて，ノードの出現回数も記録しておく．これは後の問合せ最適化に用いることを想定している．図 4 に実際の *DG* を S-DHT に格納した例を示す．例えば，b ノードは子要素とする d と g を持ち，それぞれ一つずつ存在することが分かる．

3.3　検 索 処 理

C-DHT と S-DHT による検索処理方法の概要を述べる．簡単のため，/,//だけを含む単純パス式の場合を説明する．与えられた検索式は q=|e_1|e_2|$\cdots$|e_n，e_i は要素名，|はロケーションステップの分離記号で，/ または//である．

まず，検索式の末端の要素名 e_n を手がかりに，テキストノードを直接含む情報を C-DHT によって探索する．続いて e_n をルートとする部分 XML データをたどるために，S-DHT の情報を手がかりにパスを拡張し，再度 C-DHT を参照する．具体的には，S-DHT から e_n の子要素である要素名を取得し，それをキーとして再度 C-DHT を参照することで，テキストを得る．これを部分 XML データの再構築が完了するまで繰り返す．

上記の検索処理アルゴリズムを用いて，図 5 に単純な例として，検索式 q=//c を処理する過程を示す．

① 検索式の末尾要素である c をキーに C-DHT を検索する．ここで，c を要素名に持つテキスト（もしくは葉）ノードが得られるが，検索式とパス式がマッチするかどうかを確認し，残ったものが検索結果の一部を構成するノードとなる．

② 続いて，同様に c をキーに S-DHT を検索する．ここでは，*DG* における c ノードの情報が得られ，子ノードにはテキストノードか f ノードであることが分かる．

③ そこで検索式を q=//c/f に書き換え，f をキーに再度 C-DHT を検索し，テキストノードを得る．

④ 最後に，f をキーに S-DHT を検索すると，子ノードにはテキストノードしかないことが分かるので，検索を終了する．

このアルゴリズムは，検索対象が単純なテキスト値しか含まない場合や，木構造上で葉ノードに近い部分の場合には極めて効率が良い．反面，結果がある程度の大きさを持つ部分木の場合は効率が悪化する．これに対処する方策として，部分 XML データのサイズがある程度の大きさを持つと見積もられる場合，C-DHT の内容から直接にピア ID を得て，データを保持しているピアから直接に部分 XML データを取得方法が考えられる．どちらのアルゴリズムを選択するかは，S-DHT に格納されている子ノードの統計情報など利用して推測することが可能であると考えられる．

3.4　本手法の問題点

上記の手法を実験により評価したところ良好な検索性能を得た．しかし，要素名をハッシュキーとしてデータを配置するため，同一要素名をもつノードはすべて同じピアへ格納される．したがって，大規模な P2P ネットワークにおいて，大きなデータを格納した場合，データが特定のピアに偏って配置されてしまう．

4.　拡張可能ハッシュ法に基づくストレージ負荷分散

本章では，XML データを DHT に格納する際に，各ピア間にデータ量の偏りを解消するための仕組みを説明する．第 3 章で述べた問題を解決するため，ここでは本論文で提案する，拡張可能ハッシュ法に基づく（ストレージ）負荷分散手法について述べる．

4.1　負荷分散方式

拡張可能ハッシュ法を利用した負荷分散方式について述べる．まず，各ピアはあらかじめ決められたサイズのデータ領域を複数持つものとする．これがデータページの役割を果たす．(以降

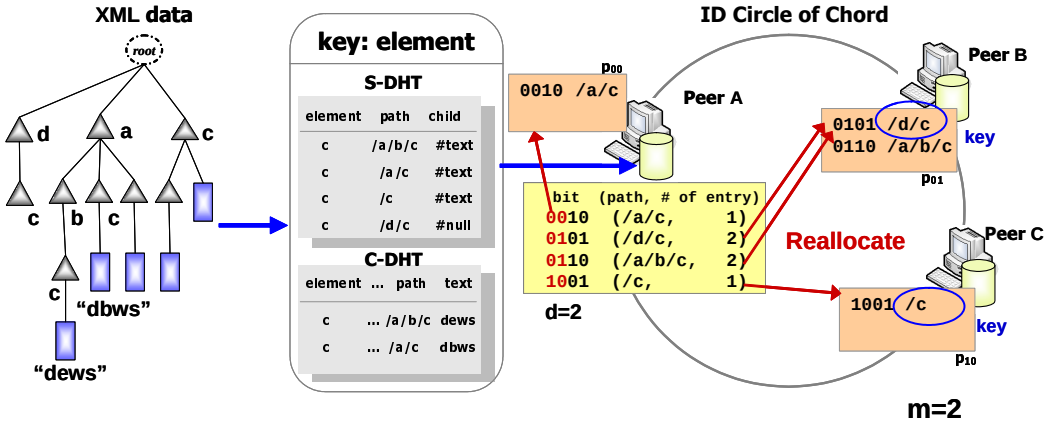


図 6　拡張可能ハッシュ法によるストレージの負荷分散
Fig. 6　Reallocating data by extendible hashing

では簡単のため，単に「ページ」と呼ぶ.）ディレクトリからポイントされるデータページは，ピアをまたがっても良い点が通常の拡張可能ハッシュとは異なる．初期状態では，ピアはディレクトリページとデータページを一つずつ持つ．

提案手法に基づき，要素 e に関するデータが DHT に格納されるとき，実際には e に該当するピアにデータが格納される．この場合，まず e の根からの絶対パス式 p_e からハッシュ関数 f によって n ビットのビット列を生成し，それに基づき拡張可能ハッシュへの挿入を行う．

同様の処理が繰り返され，データページが溢れたとき，データページの分割が行われる．具体的には，インデックスの先頭の 1 ビットが “0” か “1” かによってデータエントリを分割する．このとき，”0” に相当する（先頭の）ページはそのピアに保持するが，”1” に相当するデータページを他のピアに配分することで，ストレージ負荷の分散を図る．具体的には，データページの先頭のエントリが持つ絶対パス式をキーとして C-DHT のピアを探索する．もしそのピアが過負荷でさらなるデータを格納できない場合は，二番目以降のエントリのパス式を順に試す．データを格納できるピアを発見すると，ページを転送するとともに，ディレクトリのエントリに，キーとして与えたパス式とデータページ内のエントリ数を記録する．後者は，実際にリモートのピアにあるデータページを参照せずページ溢れを検出するために必要である．以下，同様の手順でデータページの分割を行うことで，ストレージ負荷を均衡化することができる．

図 6 に DHT に格納されているデータを，上述した手法に基づいて，他ピアへ配置する例を示す．ピア A が保持している DHT の一部は，要素名 c をキーとしたものである，データページの領域サイズは $m = 2$ である．要素名 c ハッシュキーとするデータの中では，パス式が/a/c, /d/c, /a/b/c, /c がある．この場合，パス式がデータページの領域サイズを超えているため，ページの分割が行われ，他ピアへ再配置する.（この例では，パス式/d/c, /c をハッシュキーとしたページがピア B, ピア C へ再配分される.）

4.2　検索処理

検索処理は，基本的に 3.3 節で説明した処理手順に基づく

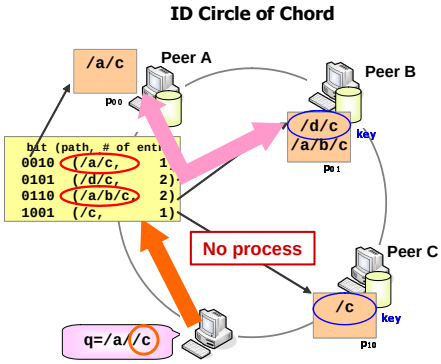


図 7　検索式 q=/a//c の処理過程
Fig. 7　Query Processing while data is reallocated by extendible hashing

が，拡張可能ハッシュ法によって他ピアに配置されたデータを取得する手順が増えるところが異なる．与えられた検索式 q=|e_1|e_2|$\cdots$|e_n を処理する際の検索処理手順を説明する．ここで|はロケーションステップの分離記号であり，/か//のいずれかである．

まず e_n を用いて 3.3 節で説明した処理手順で，e_n に関するデータを保持しているピア p を特定する．次に，p のディレクトリの各エントリ（e_n の根からの絶対パス式）から，q を満たすものを選択し，これらのパス式をキーとして，実際のデータページを取得する．

図 7 に単純な例として，検索式 q=/a//c を上述した検索処理手順の例を示す．この例では末端要素 c を用いて C-DHT を探索し，ピア A を特定する．続くピア A のディレクトリからすべてデータページへの参照を取得し，各データページに格納されているパス式から，q を満たすもの（この例では/a/c, /a/b/c）を得る．/a/c に関するものはピア A が保持しているため，直ちにピア A から解を取得する．/a/b/c では/d/c をハッシュキーとしてピア B に再配分されているため，/d/c をハッシュキーとして，ピア B を特定し．ピア B から解を得る．

5. 実　　験

上記手法の有効性を検証するため，実験による評価を行った．具体的には，第 3 章で述べた手法と，第 4 章で述べた手法，関連研究の XP2P を用いて，それぞれ以下の項目について比較する：①P2P ネットワークで各ピアが持つデータ量，②検索処理するため，必要なメッセージ数．なお，XP2P の概要については，文献 [11] および第 6 章を参照されたい．

5.1 実 験 環 境

実験で用いた計算機スペックは表 1 に示す．ネットワーク環境は，Overlay Weaver [12] が提供するネットワークエミュレータ owemu を用いて多数のピアを仮想的に生成して，P2P ネットワークを構築した．

実験用データセットとして，XMark [13] による人工データを用いた．このデータセットのサイズは 5MB で，タグ種類は 83 個であった．また，検索対象となるテキスト情報，すなわち C-DHT のデータ総数は 78837 件であり，検索処理の手がかりとして，*DG* 情報から生成した S-DHT のデータ総数は 517 件であった．

表 1　実 験 環 境
Table 1　System Configuration

CPU	AMD Opteron^{TM}2.4GHz
メモリ	16GBytes
OS	SunOS 5.10
ランタイム	JRE 1.5.0_09

5.2 ネットワークにおけるデータ量の分布

実験方法を説明する．基本的には，XML のデータセットのサイズを固定し，①P2P ネットワークの規模（ピア数 N）と，②拡張可能ハッシュ法のデータページ領域のサイズ m を変化させた．その際，上記の方式を用いたそれぞれの場合に，P2P ネットワーク上のデータ量の分布を測定した．実験結果は，図 8，図 9，図 10 に示す．それぞれはピア数 N を 100, 200, 300 個と，データページサイズ m=2, 4, 8 にさせた場合の実験結果である．

N=100 のとき，要素名をハッシュキーとした場合では，あるピアが約 15800 件のデータ量を持ち，四つのピアが約 8000 件のデータ量を持っていた．これで 1/20 のピアが全体のデータ量の 2/3 を保持していることが分かった．また，何も持たないピアは 3/4（75 個）であった．

これに対して，拡張可能ハッシュ法による方式では，データページの領域サイズ $m = 8$ の時，最大データ量（約 8000 件）を持つピアがわずか一つであり，1/3 のピアが全データ量の 1/2 を持っている．何も持たないピアは 59 個であった．$m = 4$ にした場合では，最大データ量（約 6000 件）を持つピアは一つであり，1/3 のピアが全データ量の 1/3 を持っている．何も持たないピアは半数以下（47 個）に低減することができた．また，$m = 2$ にした場合では，何も持たないピアをさらに約 1/3（38 個）に減らすことができた．一方，XP2P では一意のパス

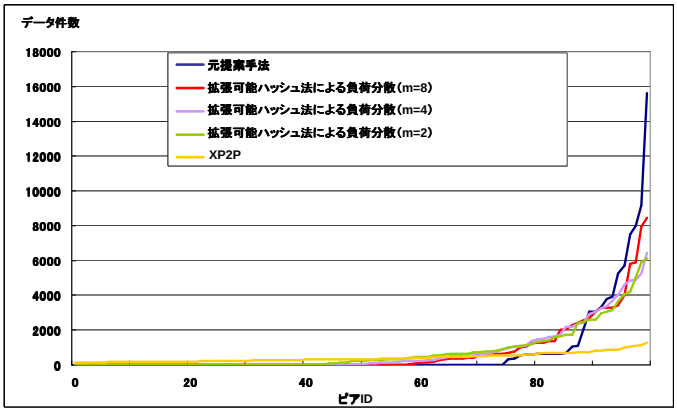


図 8　実験結果 - ピア数：100
Fig. 8　Experiment Result: 100 peers

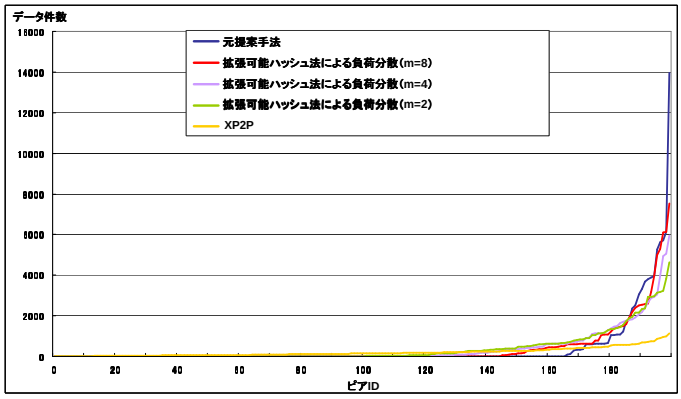


図 9　実験結果 - ピア数：200
Fig. 9　Experiment Result: 200 peers

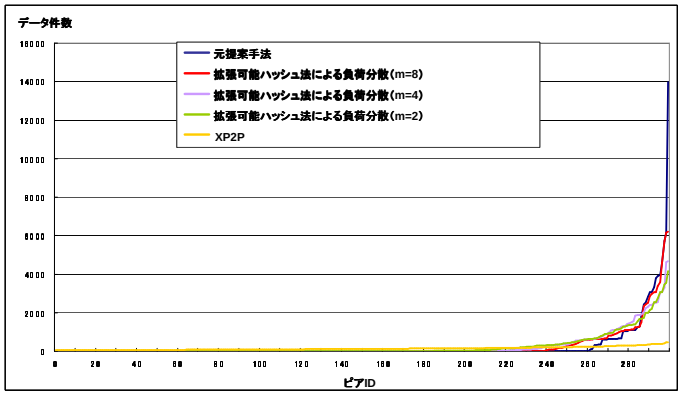


図 10　実験結果 - ピア数：300
Fig. 10　Experiment Result: 300 peers

式をハッシュキーとし，得られた値が重複することが少ないため，すべてのピアへ均等に配置された．

続く N=200, 300 個にした場合でも，ほぼ同じような傾向を示しており，データページの領域サイズ m を小さくすると，拡張可能ハッシュ法によるストレージの負荷分散の効果が顕著となった．

5.3 検索処理性能

本節では，検索処理の性能を比較する．具体的には，前節で構築した P2P ネットワークを用いて検索処理に必要なメッセー

表 2　ベンチマーククエリ
Table 2　Benchmark of Query

項目		ベンチマーククエリ
child 軸	Q_1	/site/people/person/name
	Q_2	/site/regions/asia/item/description/text/keyword
descendant 軸	Q_3	//asia
	Q_4	//closed_auctions

ジ数を測定した．本研究では簡単のため，検索式を *child* 軸 (/) および *descendant* 軸 (//) のみを含むものに限定した．実験に利用した検索式は表 2 に示す．

図 11, 図 12 に *child* 軸の検索式 Q_1, Q_2 の処理結果を示す．まず，Q_1 では，元データでは要素 `name` の子要素がテキスト値 (`#text`) だけであるため，要素名をハッシュキーとする手法と，拡張可能ハッシュ法による方式とも少ないネットワーク参照回数で済ませることができる．検索処理に必要なメッセージ数に大きな差はない．Q_2 では，元データに要素 `keyword` が子要素 `emph` と `bold`, `#text` を持つため，拡張可能ハッシュ法による方式ではより多くのデータページを参照が必要するため，より多くのメッセージ数が必要である．また，拡張可能ハッシュ法による方式では，データページの領域サイズ m によって検索処理効率が異なる．また，XP2P では，再帰的処理が必要なため，上記いずれの方式でも XP2P より優れた性能を得られることが分かった．

図 13, 図 14 に *descendant* 軸の検索式 Q_3, Q_4 の処理結果を示す．いずれの検索式も要素名をハッシュキーとする方式が拡張可能ハッシュ法による方式より効率的である．これは *descendant* 軸の検索処理には，要素名をハッシュキーとする方式の場合では，検索式の末端要素でネットワークへ 1 回だけ参照することで済む．一方，拡張可能ハッシュ法による方式ではその末端要素に関するパス式を，すべての文字列のマッチ処理をしなければならない．このため，拡張可能ハッシュ法による方式を用いた場合に検索処理に必要なデータページ参照回数が多く，メッセージ数も要素名をハッシュキーとした方式より大きく上回る．また，データページの領域サイズ m が小さくすると，前節で示したピア間のデータ量に偏り現象が緩和されるが，検索処理により多くのデータページを参照しなければならないため，メッセージ数の増加が顕著である．一方，XP2P の *descendant* 軸の検索処理には，すべてのピアにアクセスする必要があるため，メッセージ数はいずれの方式より多くなると分かった．

5.4　考　　察

上記の手法では，拡張可能ハッシュ法によるストレージ負荷分散方式を提案し，その実装および評価実験を行った．実験の結果からピアにデータ量の偏り問題が緩和された．しかしながら，本方式を用いてデータがより多くのピアへ配分されているため，検索処理に必要な参照回数が増え，ネットワークのメッセージ数が増大してしまう．また，データページの領域サイズの変化も一つの要因として，検索処理の効率に大きく影響を与えることが分かった．

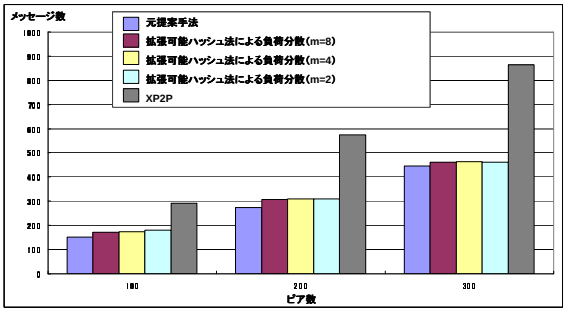


図 11　実験結果：*child* 軸のみを含む検索処理 (Q_1)
Fig. 11　Experiment Result: Q_1

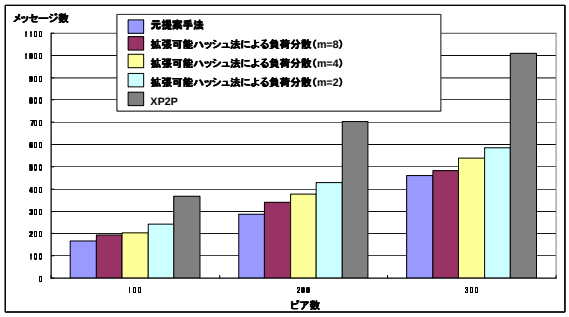


図 12　実験結果：*child* 軸のみを含む検索処理 (Q_2)
Fig. 12　Experiment Result: Q_2

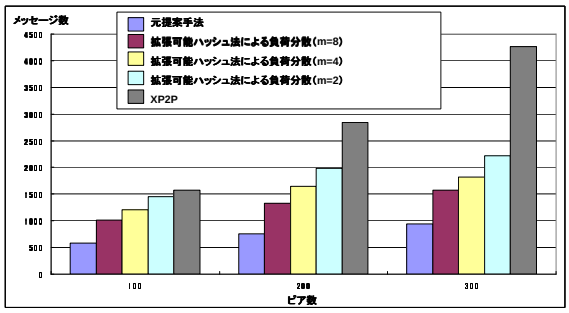


図 13　実験結果：*descendant* 軸の検索処理 (Q_3)
Fig. 13　Experiment Result: Q_3

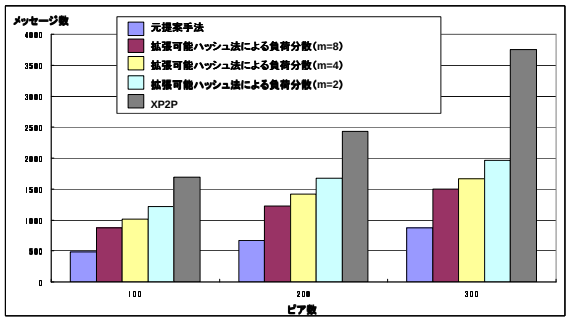


図 14　実験結果：*descendant* 軸の検索処理 (Q_4)
Fig. 14　Experiment Result: Q_4

6.　関 連 研 究

P2P 環境における XML データの検索については，Bonifati 等は XML データをフラグメント化して P2P 環境で検索する

XP2P という手法を提案している [11]．XP2P では，格納したい XML データを予め検索単位にフラグメント化しておく．フラグメントを DHT へ格納する際のハッシュキーには，対象フラグメントの親フラグメントからの相対パス式を用いる．この際，単なるハッシュ関数ではなく，Rabin によって提案されたフィンガープリンティング [14] を用いる点に特徴がある．問合せ処理では，DHT への格納キーと検索キーが正確に一致する場合は極めて効率の良い検索が可能であるが，そうでない場合は，最悪の場合にほぼ全数探索を行わなければならない．また，パス式をそのままハッシュキーに用いるため，“//” のような XML に特徴的な問合せを効率的に扱えないという問題もある．

また，P2P を用いた多様なデータの格納や検索という観点から，松下等はオブジェクトが持つ多様な特徴量をシグネチャとして表現し，シグネチャをフレームに分割した分散型シグネチャを用いることで，効率的かつ柔軟なオブジェクト検索を実現する方式を提案している [15]．また，オブジェクトの検索時および登録時の要求として，メッセージ数や応答時間を抑えたいといった要求が考えられる．Cai 等の RDFPeers は， RDF トリプルの主語，述語，目的語を，それぞれの DHT に格納することにより，RDF トリプルのための検索機構を提案している [16]．具体的には，分散ハッシュ表を用いて RDF トリプルの格納および検索を実現する分散 RDF データベースで，主語，述語，目的語のそれぞれをキーとし，RDF トリプルをその値として分散ハッシュ表に格納する手法である．すなわち，一つのトリプルを格納するには三回の探索が必要で，逆に，主語，述語，目的語のいずれかが決定できれば，それをキーとすることで，その要素を含むトリプルを検索できる．

7. おわりに

本研究では，構造型 P2P ネットワークにおいて，XML データの格納・検索方式のための負荷分散機構を提案した．XML データの要素名をハッシュキーとして DHT に格納し，拡張可能ハッシュ法を用いて，XML データのパス式をハッシュキーとして，DHT に再配置することで，各ピア間のデータ量の偏りを解消した．実験により本方式の有効性や検索処理性能について検証した．

今後の課題としては，上記した方式において，パス式の分布に偏りがある場合，パス式のプレフィックスを用いる負荷分散方式が考えられる．なお，より大規模なネットワーク環境を構築することや，多様なデータによる性能評価を行う予定であり，XQuery などの検索言語に対応する処理方式の検討などが挙げられる．

謝　辞

本研究の一部は，科学技術振興機構・戦略的創造研究推進事業 (CREST)「自律連合型基盤システムの構築」，および科学研究費補助金，萌芽研究 (#18650018)，若手研究 (B)(#17700110)，基盤研究 (B)(#15300029) により行われた．ここに記して謝意を表す．

文　献

[1] World Wide Web Consortium: Extensible Markup Language (XML) 1.0, http://www.w3.org/TR/REC-xml. W3C Recommendation 04 February. 2004.

[2] Chunhui Wu, Toshiyuki Amagasa, and Hiroyuki Kitagawa. XML Retrieval Using Structural Summary in Peer-to-Peer Environment. In *Proc. DEWS*, March 2006.

[3] 呉俊輝, 天笠俊之, 北川博之. P2P 環境における構造概要を利用した XML データの検索手法の実装について. 夏のデータベースワークショップ (DBWS2006), July 2006.

[4] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hary Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. ACM SIGCOMM*, pp. 149–160, August 2001.

[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A Scalable Content-Addressable Network. In *Proc. ACM SIGCOMM*, pp. 161–172, August 2001.

[6] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-Resilient Wide-Ares Location and Routing. Technical Report CSD-01-1141, University of California, Berkly, 2001.

[7] Roy Goldman and Jennifer Widom. DataGuides: Query Formulation and Optimization in Semistructured Databases. In *Proc. VLDB*, pp. 436–445, August 1997.

[8] Ronald Fagin, Jurg Nievergelt, Nicholas Pippenger, and H. Raymond Strong. Extendible Hashing - A Fast Access Method for Dynamic Files. *ACM Transactions on Database Systems (TODS)*, Vol. 4, No. 3, pp. 315–344, September 1979.

[9] Thomas Schwentick. XPath Query Containment. *ACM SIGMOD Record*, Vol. 33, No. 1, pp. 101–109, March 2004.

[10] Igor Tatarinov, Stratis Viglas, Kevin Beyer, Jayavel Shanmugasundaram, Eugene Shekita, and Chun Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *Proc. SIGMOD*, pp. 204–215, June 2002.

[11] Angela Bonifati, Ugo Matrangolo, Alfredo Cuzzocrea, and Mayank Jain. XPath Lookup Quries in P2P Networks. In *Proc. WIDM*, pp. 48–55, November 2004.

[12] 首藤一幸, 独立行政法人 産業技術総合研究所. http://overlayweaver.sourceforge.net, 2006.

[13] XMark - An XML Benchmark Project: http://monetdb.cwi.nl/xml, 2003.

[14] Michael O. Rabin. Fingerprinting by Random Polynomials. Technical Report CRCT TR-15-81, Harvard University, 1981.

[15] 松下亮, 北川博之, 石川佳治. P2P 環境におけるシグネチャを用いたオブジェクト検索方式. 情報処理学会論文誌：データベース, Vol. 44, No. SIG 12(TOD 19), pp. 139–149, September 2003.

[16] Min Cai and Martin Frank. RDFPeers: A Scalable Distributed RDF Repository based on A Structured Peer-to-Peer Network. In *Proc. WWW*, pp. 650–657, May 2004.