DEWS2003 8-C-03

P2P 環境におけるシグネチャを用いたオブジェクト検索

松下　亮†　北川　博之††　石川　佳治††

† 筑波大学理工学研究科　〒 305–8573 茨城県つくば市天王台 1–1–1
†† 筑波大学電子・情報工学系　〒 305–8573 茨城県つくば市天王台 1–1–1
E-mail: †ryo@kde.is.tsukuba.ac.jp, ††{kitagawa,ishikawa}@is.tsukuba.ac.jp

あらまし　近年，計算機の高性能・低価格化とネットワークインフラの発達により P2P 技術が注目されている．グローバルな索引等を持たない P2P 環境では，オブジェクトの効率的検索をどのように実現するかが問題となる．ハッシング等を用いることでその効率化を図るアプローチがこれまでに提案されているが，検索の柔軟性に欠けるという問題点がある．本研究では，オブジェクトの特徴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する．また，シミュレーション実験により本手法の評価検討を行う．

キーワード　P2P，情報検索，シグネチャファイル

Object Retrieval Using Signatures In Peer-to-Peer Environments

Ryo MATSUSHITA†, Hiroyuki KITAGAWA††, and Yoshiharu ISHIKAWA††

† Master's Program in Science and Engineering, University of Tsukuba
Tennohdai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan
†† Institute of Information Sciences and Electronics, University of Tsukuba
Tennohdai 1–1–1, Tsukuba, Ibaraki, 305–8573 Japan
E-mail: †ryo@kde.is.tsukuba.ac.jp, ††{kitagawa,ishikawa}@is.tsukuba.ac.jp

Abstract　Peer-to-peer (P2P) technology has attracted a lot of attention in recent years. Efficient object retrieval is an important research issue in P2P environments, especially in those without centralized global indices. Although a number of hash-based basic object retrieval schemes are known to alleviate the problem, they cannot provide flexible feature-based object search. In this paper, we propose a novel object retrieval method using distributed frame sliced signatures, and evaluate its effectiveness with simulation experiments.

Key words　P2P, information retrieval, signature files

1. はじめに

近年，計算機の高性能化とネットワークインフラの発達により，Peer-to-Peer （P2P）技術が注目されている．P2P では各端末が peer node（ノード）となり，大規模な分散ネットワークを構築する．各ノードはサーバ，クライアントといった明確な区別はなく，両方の役割を担っている．P2P ネットワークの形態は，クライアント・サーバシステムを融合させたハイブリッド P2P 型と，完全な分散環境であるピュア P2P 型に分類される．ハイブリッド P2P 型は，ある種のサービスを提供するために特定のサーバが存在する．情報検索というサービスにおいてハイブリッド P2P 型では，サーバがインデックス機能を提供することで，ノードに分散する情報の共有を図ることが可能である．しかし，多数のノードがそのサーバのサービスを要求した場合には，サーバがボトルネックとなる．また，サーバが停止した場合にはサービス全体が停止してしまう．一方，ピュア P2P 型では各ノードが自律的に動作する．このため，拡張性に富みボトルネックのない処理を実現することができる．しかし，ピュア P2P 型ではグローバルなインデックス等を持つことができないため，一般に情報の共有は容易ではない．代表的なピュア P2P 型のシステムとして，ファイル共有システム Gnutella [3] が挙げられる．Gnutella ではブロードキャストを用いて，周辺のノードを巡回する方法で検索を行うため，データ検索時における通信コストが大きな問題となる．この問題を解決するため，P-Grid [1]，Chord [6]，CAN [5]，Tapestry [4] 等の手法が提案されている．しかし，これらではオブジェクト ID による完全一致検索のみが考慮されており，オブジェクトの持つ種々の特徴量による検索を直接行うことはできない．

本研究では，オブジェクトの特徴をシグネチャとして表現し，ピュア P2P 型環境のノードに分散配置されたシグネチャ情報を用いることで，多様な特徴量によるオブジェクト検索を実現する．本手法を実現するための基本アーキテクチャとして Chord の枠組みを利用し，その上にシグネチャを用いた検索機構を構築する．またシミュレーション実験により，オブジェクト検索時に要するメッセージ数，オブジェクトの追加時に要するメッセージ数，およびオフラインノードが存在する場合での検索の精度について評価検討を行う．

本稿では，まず 2 章で関連研究について述べる．次に 3 章で本研究で用いる Chord アーキテクチャについて説明する．4 章で本研究における提案手法とそのアルゴリズムについて述べ，5 章で本手法に対する評価実験について述べる．最後にまとめと今後の課題について述べる．

2. 関 連 研 究

既に述べたように，P2P 環境においてオブジェクト ID に基づく効率的な検索を実現するための方法として，P-Grid，Chord，CAN，Tapestry 等がある．

P-Grid は，仮想的な二分木をネットワーク上に構成し，二分木のルートからのパス情報を用いることで検索の効率化を計る．各ノードは二分木のリーフ部分に配置される．ルートからノードまでのパス情報はバイナリで表現されており，ノードはパス情報の各プレフィックスに対応するノードをルーティング情報として保持している．一方，データオブジェクトを一意に決定するオブジェクト ID も同様にバイナリで表現されている．オブジェクト ID による問合せが任意のノードから開始されると，オブジェクト ID とノードのパスとが完全に合致しているかどうかを判断する．完全に合致していれば，検索対象オブジェクトを当該ノードが持つこととなり，問合せ結果として開始ノードに返す．完全に合致していなければ，ルーティング情報からオブジェクト ID のプレフィックスと合致するノードへ問合せをフォワードする．この処理を繰り返すことによって検索が行われる．検索に必要なメッセージ数は O (log (N)) である．

Chord では円状のネットワーク空間を構成する．各ノードはネットワーク空間全体をカバーするようなルーティング情報を保持している．Chord での検索に必要なメッセージ数は O (log (N)) である．Chord については 3 章で詳しく説明する．

CAN ではネットワーク空間を多次元空間とみなし，空間の分割・結合を行うことによって効率化を実現する．各ノードはシステムで割り当てられた重複しない部分空間に対応し，その空間内に存在するオブジェクトを管理している．ルーティング情報として，ある次元で接している隣接空間に割り当てられたノード情報を保持している．次元数を d とした場合に，検索に必要なメッセージ数は O ($dN^{1/d}$) である．

Tapestry では，ノード ID に対するプレフィックスベースのルーティング処理を行うことで効率化を図っており，検索に必要なメッセージ数は O (log (N)) である．

また，研究 [7] では，Gnutella の幅優先探索と，Freenet [2] の深さ優先探索を組み合わせ，メッセージ数と検索の応答時間のトレードオフを実現する．この研究では，幅優先探索と深さ優先探索との組み合わせの方式をいくつか挙げている．これらの方式は，ブロードキャストのポリシーの違いに基づくものであり，各方式に対して実験，評価を行っている．しかし，いずれの方式でもノードを順次巡回することで，対象オブジェクトを探索することに変わりはない．

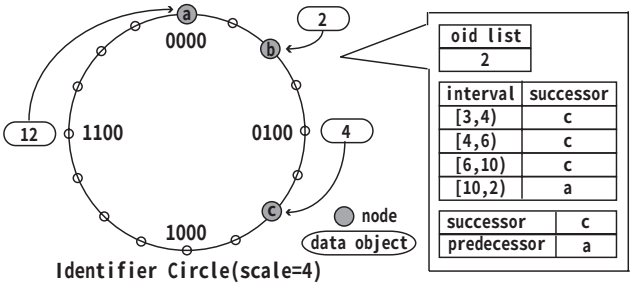

図 1　Chord アーキテクチャ

3. Chord アーキテクチャ

Chord では，ネットワーク空間全体が **ID サークル**（Identifier Circle）という円状の仮想空間として定義され，すべてのノードとすべてのデータオブジェクトはこの ID サークル上に配置される（図 1）．Chord のネットワーク空間の大きさは $scale$ で表現され，最大 2^{scale} 個のノードから構成される．Chord ではハッシュ関数を用いることで，ノードに対して $scale$ ビットの**ノード ID**（nid），データオブジェクトに対しても $scale$ ビットの**オブジェクト ID**（oid）がそれぞれ与えられる．各ノードはノード ID を基に ID サークル上に配置される．また各データオブジェクトは，オブジェクト ID から時計回りに ID サークルを辿り，ノードへ割り当てられる．

各ノードは割り当てられたオブジェクト ID のリスト，ルーティング情報，前後に位置するノード情報（$successor$，$predecessor$）を保持している．ルーティング情報には，検索対象のオブジェクト ID が与えられた時に当該ノードにそのオブジェクトが存在しない場合に対して，次にその問合せをフォワードすべきノードの情報が含まれている．この情報はノードごとに $scale$ 個存在し，各 $interval$ に対応するノード ($successor\ node$ と呼ばれる) が示されている．$interval$ とは次式で定義される区間のことである．

$$interval = [(nid + 2^{k-1}) mod\ 2^{scale}, (nid + 2^k) mod\ 2^{scale})$$
$$(1 \leqq k \leqq scale)$$

したがって，各 $interval$ の大きさは時計回りに順に 2^{k-1} となっている．これらのルーティング情報を持つことで，各ノードは ID サークル全体をカバーしていることになり，任意の問合せに対して適切なルーティング処理を行うことができる．

3.1 検 索 処 理

次に，Chord アーキテクチャでの検索処理についてより詳しく説明する．Chord ではオブジェクト ID に基づく検索のみが考慮されている．このため，まず問合せでは獲得したいオブジェクト ID を指定する．問合せは任意のノードから開始する

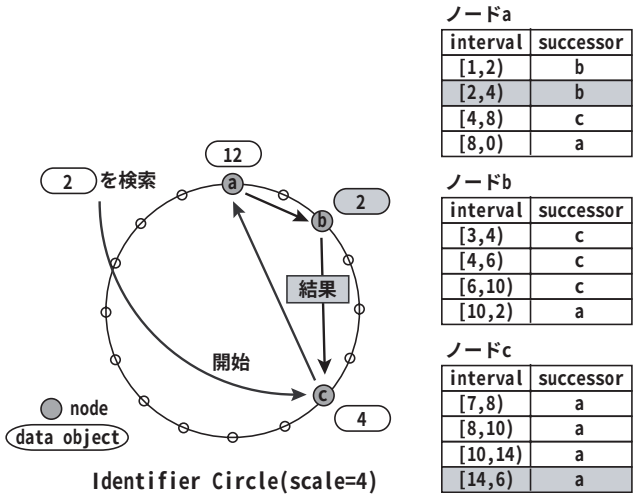

図 2　Chord における検索処理

ことができる．問合せがノードへ渡されると，ノードは割り当てられたオブジェクト ID のリストから，その問合せのオブジェクト ID を持つデータオブジェクトが存在するかどうかを判定する．存在する場合には，そのデータオブジェクトを返すことにより検索を終える．存在しない場合には，ルーティング情報を見ることで，次にフォワードすべきノードを決定しそのノードへ問合せを送る．この一連の判定処理を繰り返すことで，オブジェクト ID による検索を実現している．$N = 2^{scale}$ 個のノードを想定した場合，問合せがフォワードされる度に検索空間を半分に絞っていくことから，あるデータオブジェクトを検索するために要するメッセージ数は $O(log(N))$ となる．

具体的な検索の例として，図 2 を用いて説明する．問合せとして，オブジェクト ID が'2' のオブジェクトを検索することを考える．まず，ノード c より問合せが開始されたものとする．ノード c では問合せのオブジェクト ID を持つオブジェクトは存在しないため，ルーティング情報の中から'2' を *interval* に含む [14, 6) の *successor* であるノード a へ問合せをフォワードする．ノード a でも同様に，問合せのオブジェクト ID を持つオブジェクトは存在しないため，ルーティング情報の中から'1' を *interval* に含む [2,4) の *successor* であるノード b へ問合せをフォワードする．ノード b では，問合せ条件に合致するデータオブジェクトが存在するため，開始ノード c へ結果を返し，検索は終了する．

4. 提 案 手 法

本研究では，Chord アーキテクチャに基づく P2P ネットワークのノード上に，シグネチャ情報を分散配置することで，多様な特徴量に基づくオブジェクト検索の実現を計る．

検索対象のデータオブジェクトはユーザが任意のノードに配置し，シグネチャ情報を含むインデックスエントリを分散配置する．インデックスエントリの配置処理，および検索処理は Chord の枠組みを利用する．各データオブジェクトはノード単位で管理されるため，データオブジェクトのノード内での ID とそれを格納するノード ID のペアが，データオブジェクトを一意的に決定するキーとなる．Chord のオブジェクト ID（*oid*）と本提案手法のデータオブジェクトの ID を明確に区別するため，本提案手法におけるデータオブジェクトの ID のことを**ローカル ID**（*lid*）と呼ぶことにする．

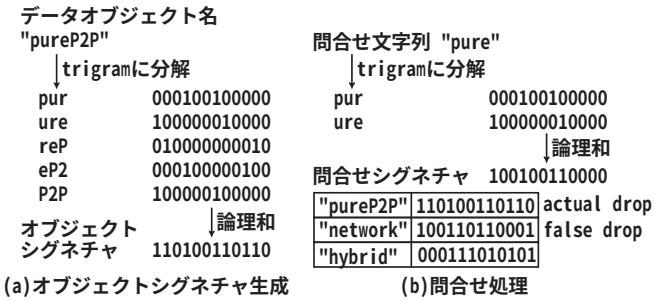

図 3　シグネチャによる部分一致検索

4.1 シグネチャ

シグネチャは，個々のデータオブジェクトから生成される固定長のビット列であり，オブジェクトの特徴量を表現するものである．一般的に，シグネチャは任意の数の特徴量を表現することができるが，特徴量が多くなればなるほどシグネチャ長を大きくする必要がある．図 3（a）は，データオブジェクト名から生成した *trigram* を特徴量とした場合の例を示している．**オブジェクトシグネチャ**の生成にはスーパーインポーズドコーディングを用いている．これは各特徴量をハッシングしてシグネチャ長の**要素シグネチャ**を生成し，さらにそれらの論理和をオブジェクトシグネチャとするものである．また，シグネチャに含まれる'1' の数のことをシグネチャのウェイトと呼ぶ．

問合せに関しては，オブジェクトシグネチャと同様にして**問合せシグネチャ**を作成する．オブジェクトシグネチャと問合せシグネチャの論理積をとったものが，問合せシグネチャと一致するときにそのオブジェクトは問合せ条件を満たす解の候補となる．この解の候補のことをドロップといい，この中で実際に正解となるものを**アクチュアルドロップ**，そうでないものを**フォールスドロップ**という．この判定処理のことをフォールスドロップレゾリューションという（図 3（b））．また，解の候補がフォールスドロップとなる確率をフォールスドロップ確率といい，以下の式で与えられる．

$$Fd = \frac{\text{フォールスドロップ数}}{\text{問合せ条件を満たさないデータオブジェクト数}}$$

フォールスドロップ確率はシグネチャの検索精度を測る尺度となる．

4.2 ロ ケ ー タ

データオブジェクトから生成したオブジェクトシグネチャを図 4 に示すように分割フレーム数 *slice* 個の**フレームシグネチャ**に分割する．Chord との整合上，*slice* は 2 の指数乗であるとする．特に，*slice* がシグネチャ長と一致する場合を**ビットスライス構成**と呼ぶ．次に，フレーム番号をバイナリ表現に変換したビット列と当該フレームシグネチャを結合し，合成シグネチャを得る．

本提案手法では，合成シグネチャから長さが *scale* ビットの**ロケータ**を生成する．ロケータは ID サークル上へのインデッ

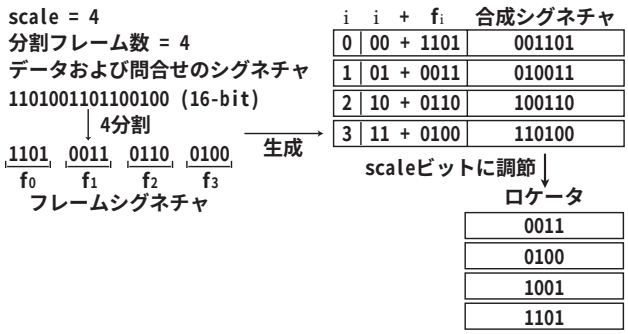

図 4　ロケータ生成

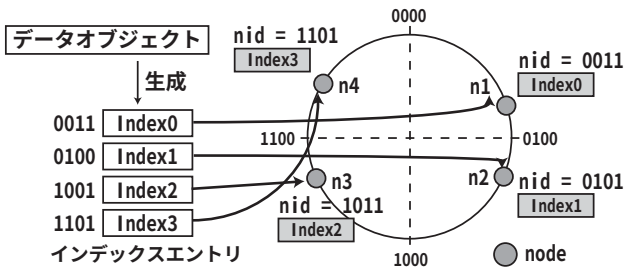

図 5　インデックスエントリの配置

クスエントリの配置，および検索処理で利用する．ロケータの生成方法は次の通りである．

[**ケース 1**] 合成シグネチャ長が *scale* の場合，合成シグネチャ自身をロケータとする．

[**ケース 2**] 合成シグネチャ長が *scale* より大きい場合，先頭から *scale* 番目までのプレフィックスをロケータとする．

[**ケース 3**] 合成シグネチャ長が *scale* より小さい場合，'0' を合成シグネチャに追加することで長さを調節し，ロケータとする．

4.3　インデックスエントリとその配置方法

次に**インデックスエントリ**の生成，およびその配置方法について述べる．インデックスエントリは，当該データオブジェクトのローカル ID，それを格納したノード ID，フレーム番号，およびそのフレームシグネチャから構成される．このとき，フレームシグネチャがすべて'0' のものに対しては，インデックスエントリは生成しない．ビットスライス構成の場合は，インデックスエントリ中のフレームシグネチャは常に'1' なので保持する必要はない．

インデックスエントリの配置方法について説明する．各インデックスエントリ中のフレーム番号とフレームシグネチャから 4.2 節で述べた方法でロケータを生成する．各インデックスエントリはこのロケータをオブジェクト ID とみなして，3 章で述べた Chord のアルゴリズムに従い，適当なノードへ配置される（図 5）．このとき，インデックスエントリ毎に個別の配置処理を行うのではなく，ルーティング情報のノード単位でインデックスエントリをまとめ，配置処理を行う．この配置方法では，個別に配置処理を行った場合に起こりうるような，ノードが複数のメッセージを処理することを避けることができる．

4.4　オブジェクト検索

オブジェクト検索時は，問合せ条件として与えられた特徴量から，問合せシグネチャ，フレームシグネチャ，合成シグネチャ，ロケータを順次生成する．この際，フレームシグネチャがすべて'0' で構成されるものに対してはロケータを生成しない．さらに，ロケータ単位で**部分問合せエントリ**を生成する．部分問合せエントリはロケータ，フレーム番号，フレームシグネチャから構成されるエントリである．問合せはフレームシグネチャを順次照会することで実行される．

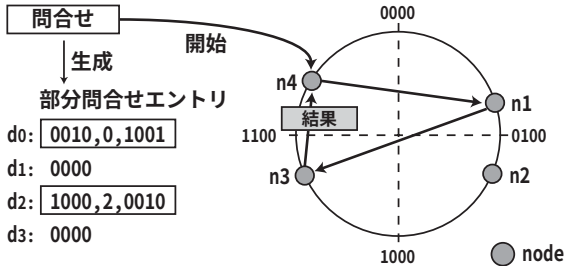

図 6　オブジェクト検索

```
1.  foreach QE in [部分問合せエントリ]
2.      検索対象ロケータ集合をQE中のロケータから計算する
3.      解候補集合Hit ← Φ
4.      foreach Loc in [検索対象ロケータ集合]
5.          Ent ← Locの配置インデックスエントリを獲得し条件判定を行う
6.          Hit ← Hit ∪ Ent
7.      if(最初の部分問合せエントリ)
8.          Ans ← Hit
9.      else  /*2回目以降の部分問合せエントリ*/
10.         Ans ← Ans ∩ Hit
11. return Ans
```

図 7　検索アルゴリズム

図 7 が検索アルゴリズムである．まず部分問合せエントリのロケータから**検索対象ロケータ集合**を生成する（2 行目）．検索対象ロケータ集合は，ロケータのフレームシグネチャ中の'0' を'0' または'1' としたすべてのビット列の集合である（したがって次式を満たす．検索対象ロケータ集合の各要素 $\wedge$(注1) ロケータ = ロケータ）．この時点では部分問合せエントリに対する検索処理は何も行われていないため，解候補集合は空集合である（3 行目）．ただし解候補集合とは，ノード ID とローカル ID のペアを要素とする集合である．次に，検索対象ロケータ集合の各要素により配置されているインデックスエントリを保持するノードに，その時点の解候補集合と部分問合せエントリを送付する．ここでは，あるロケータ *Loc* により配置されているインデックスエントリ集合のことを *Loc* の**配置インデックスエントリ**と呼ぶ．これらを受け取ったノードでは，自分の保持する配置インデックスエントリの中で，以下の 2 つの条件を満たすインデックスエントリを選択し（5 行目），その中のノード ID とローカル ID のペアを解候補集合に加える（6 行目）．

[**条件 1**] 部分問合せエントリ中のフレーム番号 = インデックスエントリ中のフレーム番号．

[**条件 2**] 部分問合せエントリ中のフレームシグネチャ $\wedge$ インデックスエントリ中のフレームシグネチャ = 部分問合せエントリ中のフレームシグネチャ．

すべての検索対象ロケータ集合中の要素について解候補集合の絞り込み処理が終了した時点で，当該部分問合せエントリによ

(注1)：'$\wedge$' はビット論理積を表す

る問合せが終了する．このあと，これまで計算された Ans と解候補集合との集合の積をとる（10 行目）．この処理を繰り返し，すべての部分問合せエントリによる解の絞り込み処理が終了した段階で，開始ノードに Ans を返す（11 行目）．

図 6 の問合せを例として説明する．まず，開始ノード n_4 に問合せが送られるものとする．問合せはノード n_4 のノード ID からの距離が最も短いロケータを持つ部分問合せエントリ d_0 から開始される．d_0 から検索対象ロケータ集合 d_0Set {'0010','0011'} が生成され，'0010' の配置インデックスエントリの有無を調べる．ノード n_4 には，'0010' の配置インデックスエントリは存在しない．このため，ルーティング情報を用いることで'0010' の配置インデックスエントリが存在するノード n_1 を辿る．このとき問合せが終了していない部分問合せエントリ $\{d_0, d_1\}$ と解候補集合を送付する．ノード n_1 では，d_0Set 中の各ロケータの配置インデックスエントリを調べることで，解候補集合を獲得する．この時点で，d_0Set 中のすべてのロケータに対する解の判定処理は終了となるため，d_0 による問合せは終了する．部分問合せエントリによる問合せが終了するごとに，これまでに得られた Ans の再計算を行う．次に d_0 と同様に，部分問合せエントリ d_1 から検索ロケータ集合 d_1Set {'1000','1010','1001','1011'} が生成され，'1000' の配置インデックスエントリの有無を調べる．ノード n_1 には，'1000' の配置インデックスエントリは存在せず，ルーティング情報を使って配置インデックスエントリの存在するノード n_3 を辿る．このとき，終了していない部分問合せエントリ $\{d_0\}$ と解集合 Ans を同時に送付する．ノード n_3 には，d_2Set 中のすべてのロケータの配置インデックスエントリが存在し，d_1 による問合せは終了する．先ほどの d_0 の終了時と同様に，Ans の再計算を行う．この段階で問合せは終了し，開始ノード n_4 へ検索結果である Ans が返される．

最終的なデータオブジェクトの取得について述べる．まず開始ノードは検索結果の Ans を受け取る．次に Ans 中の全てのデータオブジェクトを取得する．最後にフォールスドロップレゾリューションを行う．

5. 評価実験

シミュレーションに基づく本提案手法の評価実験を行った．実験では検索に必要なメッセージ数と総転送データ量，オブジェクトの追加に伴うインデックスエントリの更新のためのメッセージ数，各ノードの保持するインデックスエントリ数，およびオフラインノードが存在する場合の検索の精度を測定する．実験時の主なパラメータを表 1 に示す．データオブジェクトの特徴量の数に関しては，シグネチャ長が 2^{10} の場合で，特徴量 1 個を与えた問合せに対するフォールスドロップ確率が約 5%となるように定めた．

さらに，フォールスドロップ確率を約 5%に維持した状態で，シグネチャ長，およびデータオブジェクトより生成されたシグネチャのウェイトを変化させたパラメータ（表 2）での評価実験を行った．

表 1　主な実験パラメータ

$scale$	10
ノード数	128, 256, 512
ノード当たりのデータ数	100
データオブジェクトの特徴量の数	164

表 2　シグネチャ長とシグネチャのウェイトのパラメータ

シグネチャ長	シグネチャのウェイト
2^{10}	512
2^{11}	201
2^{12}	143

5.1 オブジェクト検索

まず最初に検索コストの評価を行う．分割フレーム数 $slice$，およびシグネチャ長 F を変化させたときの，メッセージ数と総転送データ量を測定する．このときノード数は 128 とし，問合せの特徴量を 2，4，6 と変化させて実験を行った．データオブジェクトと問合せのシグネチャはランダムに生成したものを利用し，各 50 回問合せを実行したときの平均を計算する．インデックスエントリの各要素のサイズについては nid が 6[Byte]，oid が 4[Byte] とする．フレーム番号のサイズとフレームシグネチャのサイズについては $slice$ の値によって変化する．$slice$ の値を表現するために必要なビット数を bit_slice とすると，フレーム番号に必要なサイズは（bit_slice/8）[Byte]，フレームシグネチャに必要なサイズは（F/（$slice$*8））[Byte] である．

図 8 は平均メッセージ数である．メッセージ数は，分割を行わない場合も含め，分割フレーム数が大きくなると減少している．これは，問合せシグネチャを分割したときに，解の判定を行う必要のないフレームシグネチャ（すべて'0' で構成されるフレームシグネチャ）が高い確率で出現するため，メッセージ数の削減につながっている．さらにフレームシグネチャ長も小さくなるため，検索対象ロケータ集合の要素数も小さくなり，辿らなければならないノードの数も減少する．これらの理由により，メッセージ数が大幅に削減される．

総転送データ量に関しても同様の傾向がある（図 9）．フレーム分割を行わない場合（分割フレーム数 2^0）に転送データ量が小さいのは，単一のインデックスエントリのみでデータオブジェクトのオブジェクトシグネチャ全体が得られるため，解の絞り込みが瞬時に行えることによる．分割フレーム数が 2^1 以上の付近で総転送データ量が大きくなっているのは，ノード間での中間結果が大きく，検索に必要なメッセージ数も大きいためである．

実験結果より，検索コストに関しては分割フレーム数が 2^{scale} である場合が最も効率的であることが分かる．さらに，シグネチャ長が大きい場合の方がメッセージ数が小さくなっていることを確認できる．

5.2 オブジェクト追加

次に新規にデータオブジェクトが追加される場合の，インデックスエントリの配置に必要なメッセージ数について測定する．分割フレーム数とシグネチャ長 F を変化させて実験を行った．ノード数は 128，256，512 と変化させている．配置方法は

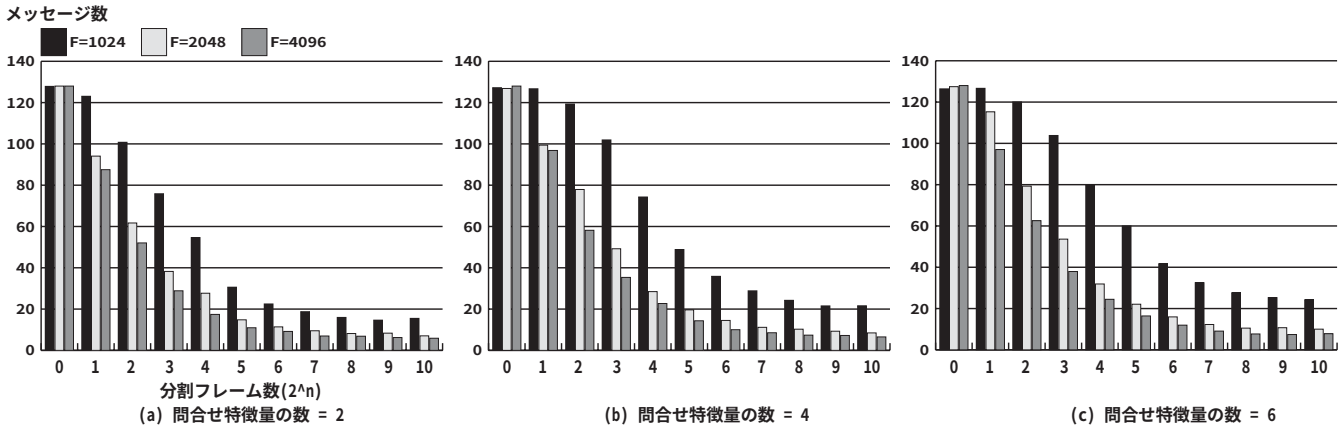

図 8　検索時における平均メッセージ数

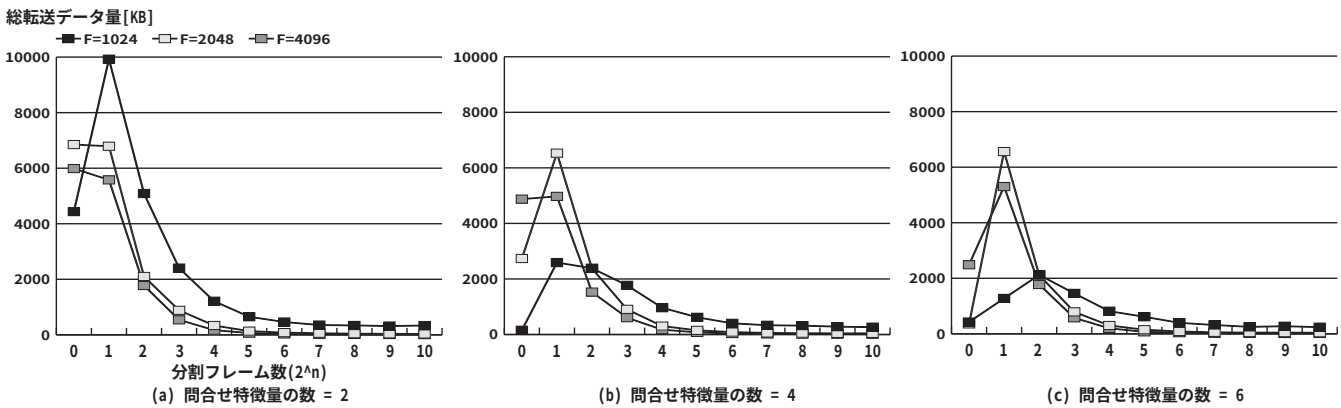

図 9　検索時における平均総データ転送量

4.3 節で述べた方法に基づいて行っている．図 10 が実験結果である．分割フレーム数が大きくなるほど，メッセージ数が大きくなっていることがわかる．この理由は，フレーム数が大きくなるほど，新たに配置する必要のあるインデックスエントリ数が多くなるからである．また，シグネチャ長を大きくした場合にはメッセージ数が小さくなっていることがわかる．これはデータオブジェクトのシグネチャのウェイトが小さくなるため，配置する必要のあるインデックスエントリの数も小さくなるからである．

5.3　検索と追加が混在する場合

オブジェクトの検索処理，およびオブジェクトの追加処理の発生頻度を考慮した場合の平均メッセージコストについて検討する．ここでは，検索処理の生起確率が p であるものとし，追加処理の生起確率が $(1-p)$ であるものとする．実験では，ノード数を 128 とし，分割シグネチャとシグネチャ長を変化させ，そのときの検索コストと追加コストにそれぞれの生起確率をかけて合計したものを平均メッセージコストとする．

図 11 が実験結果である．一般的に検索処理の方が，追加処理と比べて非常に多く行われると考えられるため，p を 0.7，0.8，0.9 と変化させた．$p = 0.7$ の場合は，分割フレーム数が 2^4 の辺りでメッセージコストが最も小さくなる．また，$p = 0.8$ の場合は，分割フレーム数が 2^5 の辺りでメッセージコストが最も小さくなっており，$p = 0.9$ の場合では，分割フレーム数が 2^5 以上であれば，最小となるメッセージコストはほぼ一定になっていることがわかる．したがって，検索処理の生起確率 p が $0.7 \leqq p \leqq 0.9$ である場合では，分割フレーム数が 2^5 付近で最小のメッセージコストが得られることがわかる．

5.4　各ノードの保持するインデックスエントリ数

1 ノード当たりのインデックスエントリ数の分布を測定した．分割するフレーム数を 2^{10} として実験を行い，1 ノード当たりのインデックスエントリの総数を計算し，5000 を一区間としてプロットする．図 12（a）がノード数 128 におけるヒストグラムである．理論上の平均値は 51200 であり，結果では平均値からの分散が大きいヒストグラムになっている．次に，ネットワーク上の総データオブジェクト数は変化させずに，ノード数を 512 まで増加させ，同様の実験を行った．図 12（b）が実験結果である．ノード数が 128 の場合と比較すると，明らかに分散が小さくなっていることがわかる．このように，一定以上のノード数が存在する状況では，ボトルネックのない処理を実現できる可能性が高い．

5.5　オフラインノードが存在する場合の検索の精度

最後に，複数のノードがオフラインの状態にあり，シグネチャの照合処理が正しく行えないと仮定した場合での，検索の精度について実験を行った．なお，Chord の枠組みではルーティング情報を動的に更新することでオフラインノードが存在する場合に対応することができるため，ルーティング情報は常に正しく利用できるものと仮定する．この実験では，ある問合せを実行し得られた結果のドロップ数からフォールスドロップ確率を

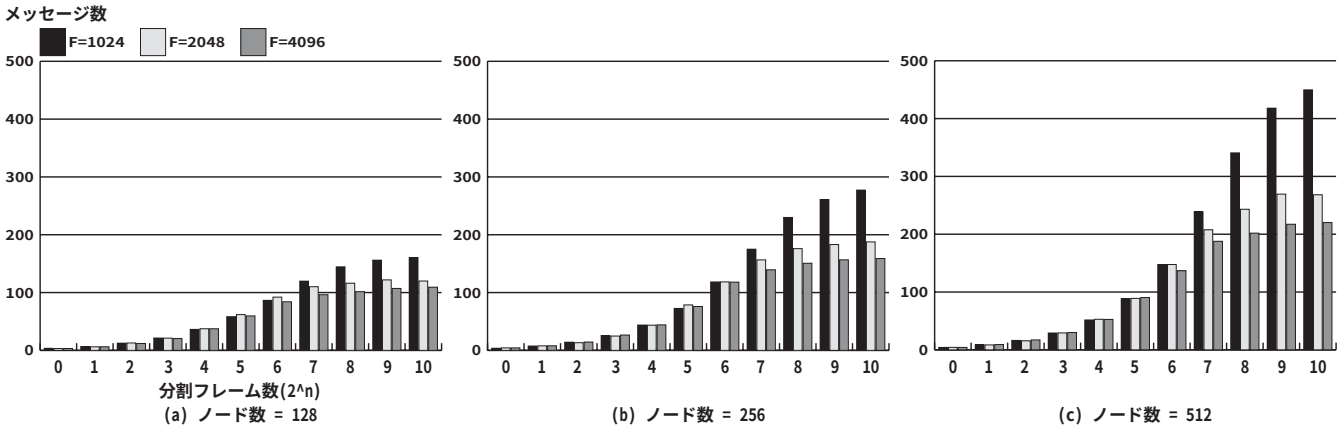

図 10　インデックスエントリ配置時における平均メッセージ数

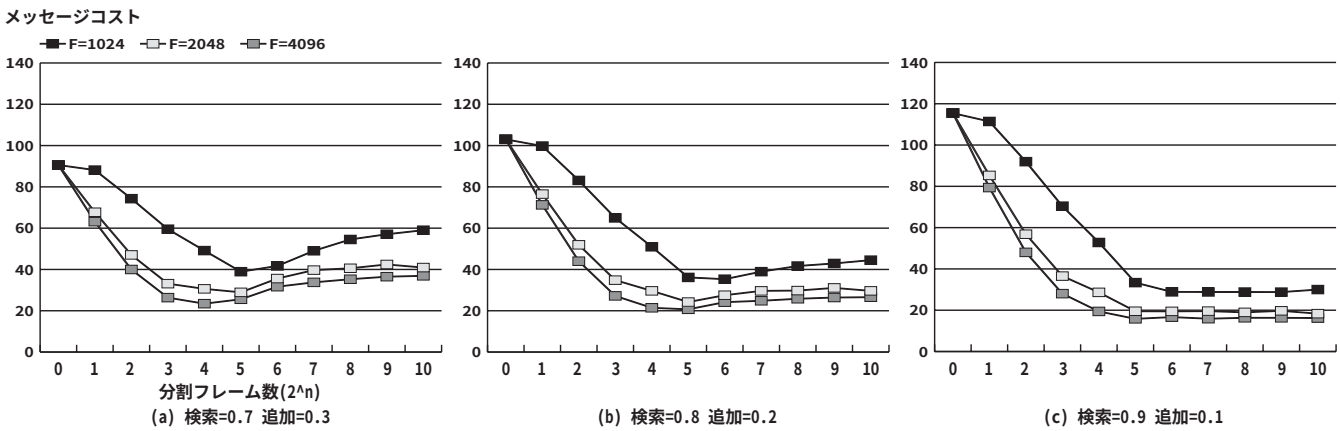

図 11　生起確率を考慮したメッセージコスト

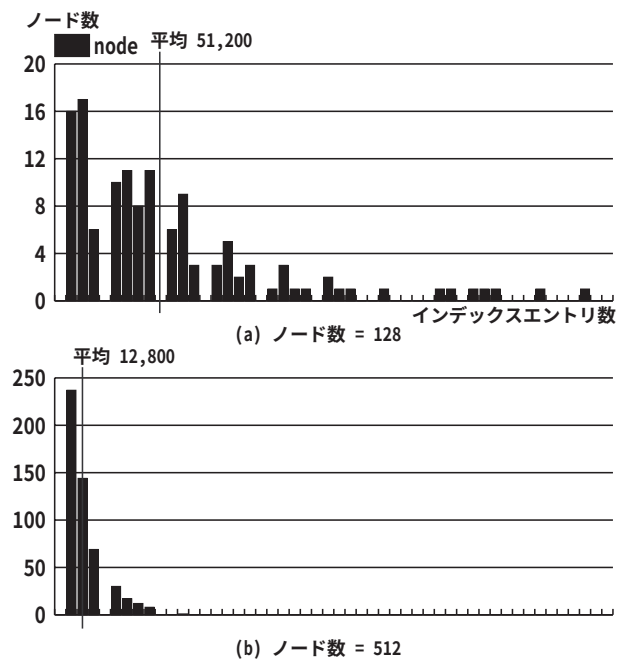

図 12　インデックスエントリ数におけるヒストグラム

計算する．この状況下では，特定のフレームシグネチャを照合することができないため，絞り込み処理を実行できず，フォールスドロップ数は大きくなることが予想される．

実験では，問合せ特徴量の数を 1 とし，分割フレーム数 $slice$ を 2^0，2^4，2^8 と変化させフォールスドロップ確率を求める．また，ノード数は 128 であるものとし，正しく照合処理が行えないノードの確率を 0%から最大 50%まで変化させる．

図 13 が実験結果である．オフラインノードの確率が大きくなるにつれて，フォールスドロップ確率も大きくなっていることを確認できるが，分割フレーム数を大きくした場合の方が，フォールスドロップ確率の増加曲線は緩やかになっている．この理由は分割フレーム数を大きくすることでフレームシグネチャのサイズが小さくなり，照合することができないシグネチャのサイズが小さくなるからである．したがって，オフラインノードが存在する状況下では分割フレーム数が大きいほど，検索の精度は高いと言える．

6. おわりに

本研究では，P2P 環境において分散配置されたシグネチャ情報を用いて，多様な特徴量を用いたオブジェクト検索を実現する手法を提案した．分割するフレーム数を大きくすることで，検索時におけるメッセージコストを削減できることを実験結果から示すことができた．また，オブジェクト検索および追加の生起確率を考慮した平均メッセージコストについても検討した．さらに，ノード当たりのインデックスエントリ数の分布についても測定し，ノード数が増加する程，偏りが減少する傾向があることを確認した．さらに，オフラインノードが存在する場合

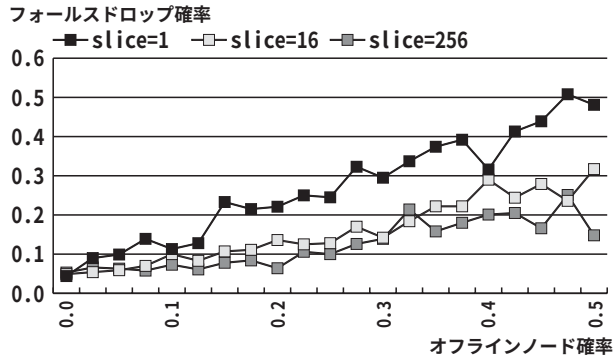

図 13　検索の精度

の検索の精度についても実験を行い，分割フレーム数が大きいほど検索精度は高いということを示した．

今後の課題として，応答時間を考慮したメッセージのフォワーディング方法の提案や，実際の計算機を用いた検索時間，追加による更新時間等の実測値を計測する必要がある．

謝　　辞

本研究の一部は，日本学術振興会科学研究費基盤研究 (B)(12480067)，奨励研究 (A)(12780183)，及び文部科学省科学研究費特定領域研究 (C)(13224008) による．

文　　献

[1] Karl Aberer, P-Grid: A Self-Organizing Access Structure for P2P Information Systems, CoopIS 2001, LNCS 2172, pp. 179-194, 2001.
[2] Freenet website. http://freenet.sourceforge.net/.
[3] Gnutella website. http://www.gnutella.com/.
[4] Kirsten Hildrum, John D.Kubiatowicz, Satish Rao, and Ben Y.Zhao, Distributed Object Location in a Dynamic Network, SPAA'02 August, 2002.
[5] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker, A Scalable Content-Addressable Network, SIGCOMM'01 August, 2001.
[6] Ion Stoica, Robert Morris, David Karger, M.Frans Kaashoek, and Hari Balakrishnan, Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, SIGCOMM'01 August, 2001.
[7] Beverly Yang, and Hector Garcia-Molina, Improving Search in Peer-to-Peer Networks, ICDCS'02 July, 2002.