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

松下 亮[†] 北川博之[††] 石川佳治[††]

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．

Signature-based Object Retrieval in Peer-to-Peer Environments

Ryo Matsushita,[†] Hiroyuki Kitagawa[††]
and Yoshiharu Ishikawa[††]

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.

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 では各計算機が 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．また，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．ハイブリッド P2P 型におけるオブジェクト検索では，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，サーバの処理能力に制約されずにノードの追加ができる等の拡張性に富み，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サーバに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[4] があげられる．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，オブジェクト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，Chord[10]，P-Grid[1]，CAN[9]，Tapestry[5] 等の手法が提案されている．

これらの手法は，以下のような特徴を持つ．

- **効率的なオブジェクト検索**
  全ノード数 N に対して $O(log(N))$ またはこれに準じた検索コストの効率的な検索方式を提供する．したがって，多数のノードが存在する環境にも適用可能である．
- **負荷の均等化**

† 筑波大学理工学研究科
Master's Program in Science and Engineering, University of Tsukuba
†† 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba

データオブジェクトは各ノードに均等に配置され，ノードあたりの負荷が均等化される．

- **ノードの追加・削除への対応**
  ノードの追加や削除に対し，自律的にノード情報を修正することで，サービスを継続することができる．

しかし，これらの手法では，オブジェクト ID というキーを用いたオブジェクト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．

一般に，オブジェクトの持つ種々の特徴量を用いた柔軟な検索を実現する手段としてシグネチャ[2),6)]が知られている．シグネチャを用いることで，多様な特徴量による検索を行うことができる．たとえば，オブジェクト名やその内容記述に対するキーワード照合や任意の部分文字列照合があげられる．

本研究では，ピュア P2P 型環境における分散シグネチャを用いたオブジェクト検索手法を提案する．本手法の特徴は次のとおりである．

1. シグネチャを用いることで，多様な特徴量による柔軟な検索に対応できる．
2. 基本となる枠組みとして Chord や P-Grid 等を利用し，かつこれらの持つ上記の特徴をシグネチャを用いた検索においても継承することができる．
3. シグネチャのフレーム分割を導入することで，適切なパラメータを選択することにより，利用環境におけるオブジェクト検索と追加の発生頻度に適合した構成をとることができる．
4. オフラインノードが存在する場合でもオブジェクト検索を継続することができる．

本論文では，基本アーキテクチャとして Chord の枠組みを利用した場合を具体的な対象とする．また，シグネチャを用いた特徴検索の例として，オブジェクト名に対する部分文字列照合を示す．さらに，シミュレーション実験により，オブジェクト検索時および追加時に要するメッセージ数とその応答時間，検索と追加が混在する場合でのメッセージ数，およびオフラインノードが存在する場合の検索の精度について評価検討を行う．

以下では，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．さらに，4 章でシグネチャの説明をし，5 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る．6 章で本手法に対する評価実験について述べ，最後にまとめと今後の課題について述べる．

2. 関連研究

すでに述べたように，効率的な検索を実現するための方法として，Chord[10)]，P-Grid[1)]，CAN[9)]，Tapestry[5)] 等がある．Chord については 3 章で詳しく説明する．これらの手法は，P2P 環境上にキーを用いた検索のための構造を導入し，適切なルーティング処理を行うことで，検索の効率化を図っている．しかし，検索はキーとしてのオブジェクト ID に基づくもののみに限られ，オブジェクトの特徴量等による柔軟な検索は不可能である．

研究 11) では，Gnutella の幅優先探索と，Freenet[3)] 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合せの方式をいくつかあげている．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．

本研究では P2P 環境におけるオブジェクト検索を実現するためにシグネチャを用いるが，並列処理を用いてシグネチャの照合処理を効率化する研究[7)] もある．この研究では，シグネチャを分割しすべての計算機に均等に割り当て，照合処理を並列化することで効率化を図るような処理方式を提案している．しかし，シグネチャの照合処理が全ノードで発生することや，シグネチャの分割や割当てが静的であることにより，この方式を P2P 環境に適応させることは非常に困難である．これに対して，本提案手法は，シグネチャの分割法と配置法を工夫することで，シグネチャの照合処理が必要なノード数を少なく抑えることが可能である．また，ノードの追加や削除に対しても，シグネチャの動的な再配置を行うことが可能である．

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 ではノードおよびデータオブジェクトを ID サークル上に均等に配置するため，ハッシュ関数を用いる．これによりノードに対して *scale* ビットの**ノード ID**（*nid*），データオブジェクトに対して *scale* ビットの**オブジェクト ID**

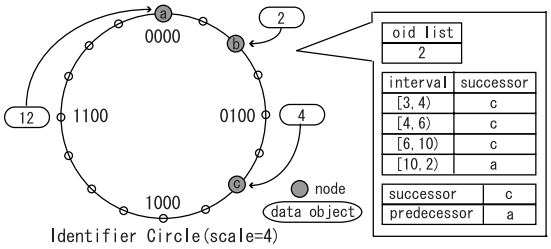

図 1 Chord アーキテクチャ
Fig. 1 Chord architecture.

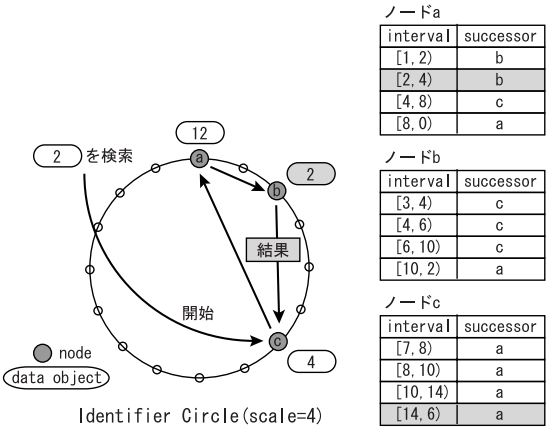

図 2 Chord における検索処理
Fig. 2 Object retrieval in Chord.

（*oid*）をそれぞれ与える．各ノードはノード ID に基づき ID サークル上に配置される．また，任意のオブジェクト ID のデータオブジェクトを ID サークル上に存在するいずれかのノードに割り当てるため，各ノードは ID 集合を持つ．ノード n のノード ID を n_{id}，その直前（時計回りの方向を順方向とする）に位置するノード n_{pre} のノード ID を n_{pre}_id とする．このとき，ノード n の ID 集合は，以下の区間 $interval_{id}$ に含まれる ID の集合のことである．

$$interval_{id} = (n_{pre}\_id, n\_id]$$

各データオブジェクトは，そのオブジェクト ID を ID 集合の要素に含む当該ノードへ割り当てられる．

各ノードは実際に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．ルーティング情報は，検索対象のオブジェクト ID のオブジェクトが当該ノードに存在しない場合に，次にその問合せをフォワードすべきノードの情報を含む．この情報は *scale* 個存在し，各 *interval* に含まれるオブジェクト ID のオブジェクトを検索する際に次にどのノード（*successor node* と呼ばれる）に検索要求をフォワードすべきかが示されている．ただし，*interval* とは次式で定義される区間のことである．

$$\begin{aligned} start &= (nid + 2^{k-1}) mod\ 2^{scale} \\ end &= (nid + 2^{k}) mod\ 2^{scale} \\ interval &= [start, end) \\ & (1 \leq k \leq scale) \end{aligned}$$

このように，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することになり，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．

Chord においてノードの追加および削除があった場合の処理について説明する．新たにノードが追加された場合には，当該ノードを含む影響を受けるノードのルーティング情報と前後に位置するノード情報を更新し，*successor* の持つオブジェクトの再配置を行う．あるノードが削除される場合には，当該ノードの持つオブジェクトを *successor* へ再配置し，影響を受けるノードのルーティング情報と前後に位置するノード情報を更新する．

次に，Chord 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そのノード自身に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 による検索を実現する．$N = 2^{scale}$ 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．

具体的な検索の例を，**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’ である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るものとする．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

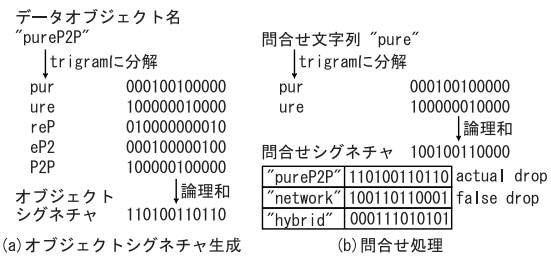

図 3 シグネチャによる部分一致検索
Fig. 3 Signature-based partial matching retrieval.

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 '2' 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．

以上，Chord ではオブジェクト ID を用いた検索については，効率的な検索が可能である．また，各ノードの ID 集合の要素数がほぼ同じになることより，負荷の均等化が図れる．さらに，ノードの追加や削除への対応もなされている．このように，1 章で述べた 3 つの特徴が実現されている．

4. 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．オブジェクトの特徴量をビット列という単純な表現方法に変換することで，特定の特徴量の存在の有無を容易に判定でき，多様な特徴量によるオブジェクト検索が可能である．

まず，各データオブジェクトから生成される**オブジェクトシグネチャ**について述べる．**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，さらに得られた要素シグネチャの各ビット列の論理和をオブジェクトシグネチャとするものである．したがって，生成されたオブジェクトシグネチャは，当該データオブジェクトの持つ特徴量をすべて表すことになる．シグネチャでは，'1' の立っているビット位置によってこれを表現する．なお，この '1' の立っているビット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（図 3 (b)）．このため，問合せシグネチャの '1' の立っているビット位置が，問合せの持つすべての特徴量を表している．オブジェクト検索では，問合せシグネチャの '1' の立っているビット位置がオブジェクトシグネチャでも '1' であるかを照合することで，当該オブジェクトが問合せの持つすべての特徴量を含む可能性があるかを判定する．したがって，オブジェクトシグネチャと問合せシグネチャの各ビット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の持つ特徴量をすべて含む可能性があり，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．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 Fd といい，以下の式で与えられる．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．

$$Fd = \frac{Num\_FalseDrop}{Num\_UnqualifiedDataObject}$$

$Num\_FalseDrop \cdots$ フォールスドロップ数
$Num\_UnqualifiedDataObject \cdots$ 問合せ条件を満たさないデータオブジェクト数

このように，*trigram* を特徴量と考えた場合には，シグネチャを用いることで，任意の文字列によるデータオブジェクト名の部分一致検索を実現できる．

5. 提 案 手 法

本研究では，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図る．最も単純な分散配置法としては，各オブジェクトシグネチャを単位として分散配置することが考えられる．この方法では，オブジェクトの追加時の処理は効率的にできるが，検索時に多数のシグネチャを参照する必要が生じる場合があり，多数のノードでの照合処理を避けるのは難しい．別の分散配置法としては，各オブジェクトシグネチャを複数の部分に分割し，部分シグネチャを単位として分散配置することが考えられる．この場合，照合時には問合せシグネチャ中の '1' の立っているビット位置を含む部分シグネチャのみを参照すればよいため検索処理の効率化を図ることができる．しかし，オブジェクト追加時には，各部分シグネチャの配置処理が必要になる．そこで，本研究では上記のオブジェクトシグネチャを単位とした分散配置法と部分シグネチャ

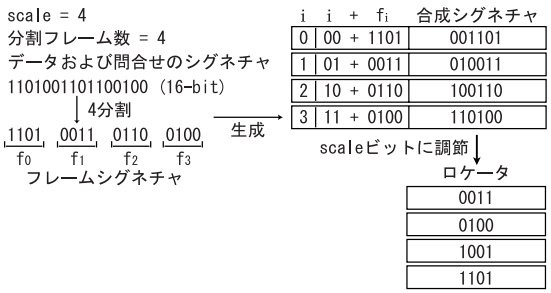

図 4 ロケータ生成
Fig. 4 Locator generation.

を単位とした分散配置法を融合し，利用環境におけるオブジェクト検索と追加の発生頻度に適合した構成が可能な手法を提案する．さらに，本提案手法は，P2P ネットワーク上でデータの効率的検索を実現する枠組みの上に構築するものであり，2 章で述べた Chord, P-Grid 等のいずれの枠組みを用いても実現することが可能である．本研究では Chord を用いた具体的な実現法を示すとともに，基本的性能評価結果を示す．

本提案手法では，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ノード内でのデータオブジェクトの ID のことを**ローカル ID**（*lid*）と呼ぶ．

5.1 ロケータ

シグネチャ情報の分散配置，および検索処理を行ううえでの前処理について説明する．まず，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[8]．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クスエントリの配置，および検索処理の際利用する．ロケータの生成方法は次のとおりである．

[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 *scale* とし，ロケータとする．

このロケータ生成方法を用いることにより，任意の分割フレーム数，およびシグネチャ長に対してロケータを生成することができる．また，このようなロケータを用いてインデックスエントリを配置することで，各ノードに対して均等にインデックスエントリを配置することができる．

なお，ここでノード n_i におけるロケータの順序を以下のように定義する．ノード n_i のノード ID を *nid*，ロケータを *loc* とした場合に，次の *ord* を計算する．

$$ord = (nid - loc) mod\ 2^{scale}$$

ロケータの順序は *ord* の昇順とする．

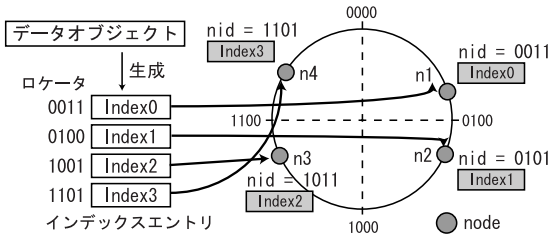

図 5 インデックスエントリの配置
Fig. 5 Index entry registration.

5.2 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，各フレームシグネチャに対して生成する．ただし，すべてのビットが ‘0’ であるフレームシグネチャに対しては，インデックスエントリは生成しない．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．

あるノードにあるデータオブジェクトが追加された場合のインデックスエントリの配置方法について説明する．本研究では，並列的にインデックスエントリの配置処理を行うことで，配置処理の応答時間を短縮する．まず，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 5.1 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 3 章で述べた Chord のアルゴリズムに従い，ロケータをオブジェクト ID と見なして，適当なノードへ配置される（**図 5**）．

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

図 6 検索アルゴリズム
Fig. 6 Retrieval algorithm.

このとき，配置処理を行う必要のある各インデックスエントリを個別に処理するのではなく，まず，データオブジェクトが追加されたノードの持つルーティング情報に基づき，次にフォワードすべきノードごとに各インデックスエントリを分類する．次に，このようにして得られた各インデックスエントリ集合を当該ノードへ送付する．同様の処理を送付されたノードで繰り返し，インデックスエントリの配置を行う．この配置方法では，各ノードが処理するインデックスエントリ配置のためのメッセージ数を最小化することができる．以下では，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集合**と呼ぶ．

5.3 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．ただし，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，これらより**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．問合せはこれらの部分問合せエントリを順次照合することで処理される．この処理は，時計回りに問合せ処理を行ううえで参照すべきインデックスエントリを持つノードを順次巡回していくことで行う．この場合，照合処理による解の絞り込みが順次行われるため，中間結果のデータ転送量が小さくなることが期待される．

図 6 が検索アルゴリズムである．まず，問合せから部分問合せエントリ集合が生成される（1 行目）．各部分問合せエントリは，問合せが発生したノードにおけるロケータの順序に従い，より小さいロケータを持つ順に処理する（2 行目）．次に，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（3 行目）．検索対象ロケータ集合は，ロケータの中にある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 ∧☆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（4 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対して，そのロケータの配置インデックスエントリ集合を保持するノードに，その時点の解候補集合，未処理の部分問合せエントリ集合，未処理の検索対象ロケータ集合を送付する．ただし，検索対象ロケータ集合からのロケータ選択は，そのノードにおけるロケータの順序に従う．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集合の中で，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（6 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（7 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 ∧ 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の各要素について解候補集合の取得処理が終了した時点で，当該部分問合せエントリに関する処理が終了する．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り，解の絞り込み処理を行う（11 行目）．この処理をすべての部分問合せエントリに対して行った後，開始ノードに Ans を返す（12 行目）．

図 7 の問合せを例として説明する．まず，ノード n_4 で問合せが発生するものとする．問合せ処理はノード n_4 におけるロケータの順序に従い，部分問合せエ

☆ ‘∧’ はビット論理積を表す．

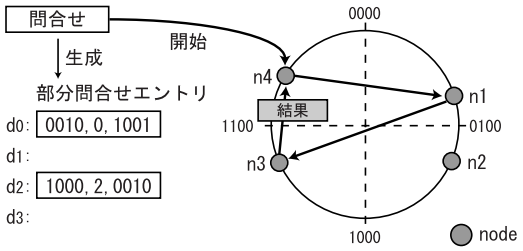

図 7 オブジェクト検索
Fig. 7 Object retrieval.

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集合の有無を調べる．ノード n_4 には，‘0010’ の配置インデックスエントリ集合は存在しない．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集合が存在するノード n_1 をたどる．このとき未処理部分問合せエントリ集合 $\{d_0, d_2\}$，未処理検索対象ロケータ集合 d_0Set，解候補集合をノード n_1 に送付する．ノード n_1 では，d_0Set 中の全ロケータ（‘0010’ と ‘0011’）の配置インデックスエントリ集合を持つため，これらを用いた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処理は終了となるため，d_0 に対する処理は終了する．次に未処理の部分問合せエントリ d_2 から検索対象ロケータ集合 d_2Set {‘1000’,‘1010’,‘1001’,‘1011’} が生成され，‘1000’ の配置インデックスエントリ集合の有無を調べる．ノード n_1 には，‘1000’ の配置インデックスエントリ集合は存在せず，ルーティング情報を使って配置インデックスエントリ集合の存在するノード n_3 をたどる．このとき，未処理部分問合せエントリ集合 $\{d_2\}$，未処理検索対象ロケータ集合 d_2Set，解集合 Ans をノード n_3 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集合が存在し，d_2 に対する処理は終了する．ここで Ans の再計算を行う．この段階でシグネチャの照合処理は終了し，開始ノード n_4 へ検索結果である Ans が返される．

開始ノード n_4 は検索結果 Ans 中の全要素に対応するデータオブジェクトを取得し，最後にフォールスドロップレゾリューションを行い，問合せに対する最終的な解を得る．

6. 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，1) 検索に必要なメッセージ数，総データ転送量，応答時間，2) オブジェクトの追加にともなうインデックスエントリの更新のためのメッセージ数と応答時間，3) 検索と追加が混在する場合でのメッセージ数，4) 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．5 章で述べたように，検索コストと追加コストの間にはトレードオフの関係があり，分割フレーム数はそのコストに大きな影響を与えるパラメータである．その関係を確認するのが，実験 1)，2)，3) の主な目的である．また，シグネチャを用いた検索の特徴の 1 つとして，すべてのフレームシグネチャを参照しなくても対象オブジェクトの絞り込みが可能な点がある．この特徴により，仮に必要なインデックスエントリを保持するノードがオフラインであっても，残りのオンラインノードの持つインデックスエントリのみで絞り込み処理を行うことが可能である．この点を確認するのが実験 4) の目的である．実験時の主なパラメータを**表 1**，**表 2** に示す．シグネチャ長およびウェイトについては，表 2 に示す 3 通りについて実験を行う．データオブジェクトの特徴量の数を 164 とした場合，特徴量 1 個に基づく問合せに対するフォールスドロップ確率はこの 3 通りのいずれの場合も約 5%となる．応答時間の測定では，ノード間で，あるメッセージを転送するために必要な転送時間を単位時間とする．その他の処理時間は，メッセージ転送に必要な時間と比較した場合に非常に小さくなると考えられるため，考慮しない．

表 1 主な実験パラメータ
Table 1 Experiment parameters.

項目	値
$scale$	10
ノード数	128, 256
ノードあたりのデータ数	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
Table 2 Signature size and weight parameters.

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

6.1 オブジェクト検索

まず最初に検索を行う際のメッセージ数，総データ転送量，応答時間の評価を行う．分割フレーム数 $slice$，およびシグネチャ長 F を変化させて実験を行う．ノード数は 128 とし，問合せの特徴量は 2，4，6 と変化させる．すでに述べたように，フレーム分割数を増加させた方が検索コストは減少することが予想される．また，シグネチャ長を大きくした場合には，問合せシグ

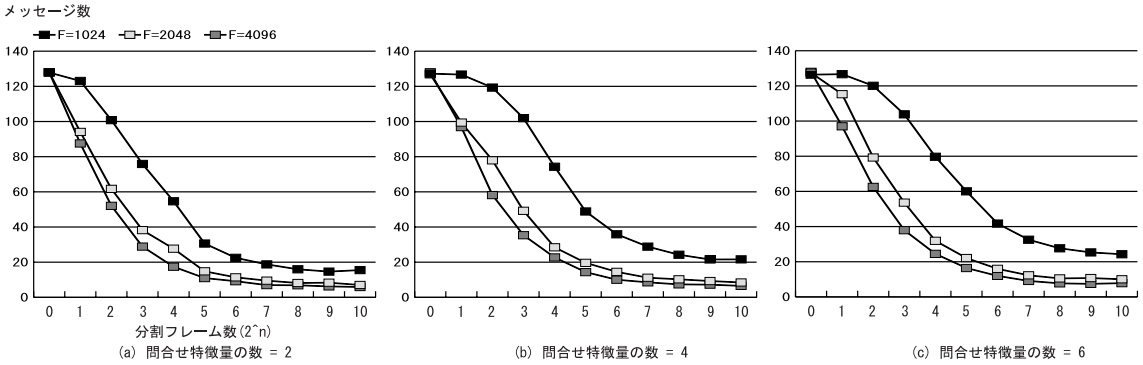

図 8 検索時の平均メッセージ数
Fig. 8 Average number of messages for retrieval.

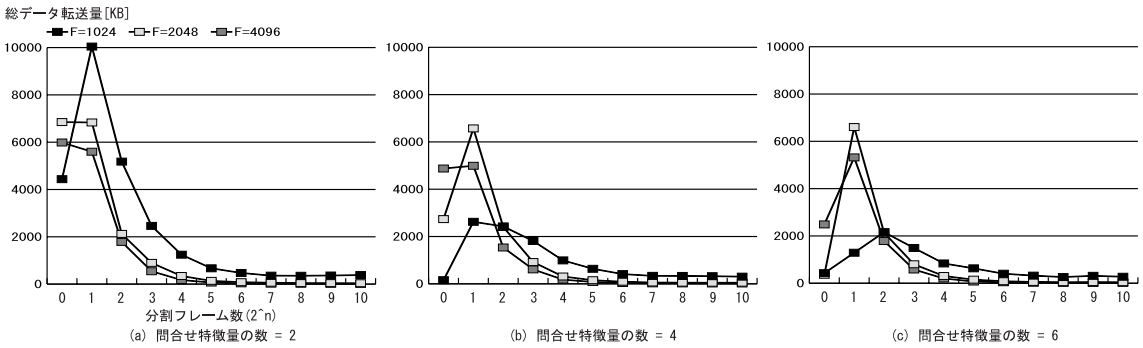

図 9 検索時の平均総データ転送量
Fig. 9 Average of transferred data size for retrieval.

ネチャのウェイトが小さくなるため，参照する必要のあるフレームシグネチャの数が減少し検索コストも小さくなることが予想される．一方，問合せ特徴量の数を大きくした場合には，問合せシグネチャのウェイトも大きくなり，検索コストは大きくなると考えられる．実験では，問合せを各 50 回実行したときの平均を測定値とする．データ転送量を測る際，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]，lid 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 とした場合，フレーム番号に必要なサイズは $\lceil bit\_slice/8 \rceil$ [Byte]，フレームシグネチャに必要なサイズは $\lceil F/(slice * 8) \rceil$ [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，たど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．

総データ転送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）にデータ転送量が小さいのは，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 以上の付近で総データ転送量が大きくなっているのは，順次巡回による十分な解の絞り込みが行えないため中間結果のデータ転送量が大きくなり，検索に必要なメッセージ数も大きくなっているためである．

応答時間について説明する．本提案手法では，各ノードを順次巡回することで検索を行っており，すべての絞り込み処理が終了した時点で結果を返す．このため応答時間は，検索時のメッセージ数（図 8）と同一の曲線となる．この場合についても分割フレーム数を大きくした場合の方が優れている．

実験結果より，検索時のメッセージ数に関しては分割フレーム数が大きくなるほど効率的であることが分

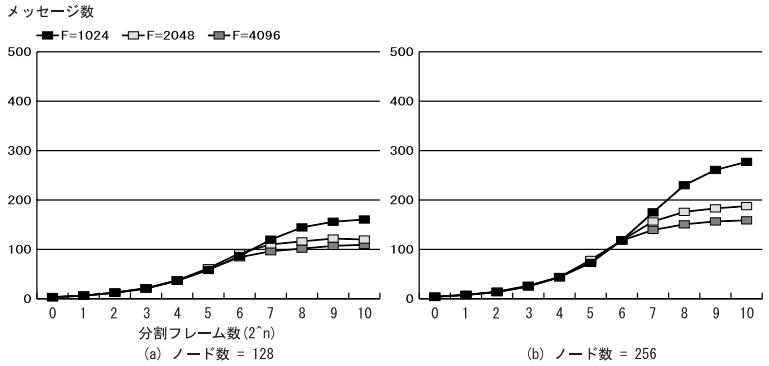

図 10 配置時の平均メッセージ数
Fig. 10 Average number of messages for index entry registration.

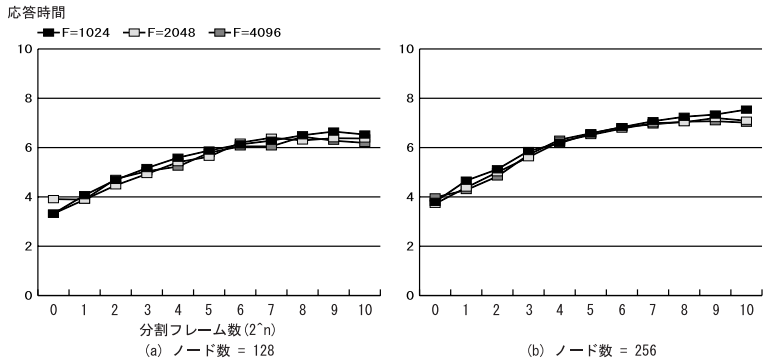

図 11 配置時の平均応答時間
Fig. 11 Average response time for index entry registration.

かる．また，シグネチャのウェイトを小さくすることで，メッセージ数を低減できることを確認できる．

6.2 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とその応答時間について測定する．実験では，分割フレーム数とシグネチャ長 F を変化させ，ノード数は 128，256 と変化させて行う．配置方法は 5.2 節で述べた方法に基づいて行う．すでに述べたように，フレーム数を増加させた方が追加の際に必要なメッセージ数は増加することが予想される．また，オブジェクト検索時と同様に，シグネチャ長を大きくした場合の方が，インデックスエントリの配置に必要なメッセージ数は小さくなることが予想される．

図 10 は平均メッセージ数である．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分かる．この理由は，分割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．シグネチャのウェイトを小さくした場合には相対的にメッセージ数が小さくなっている．これは，配置する必要のあるインデックスエントリの数が小さくなるからである．

さらに応答時間についても測定する．**図 11** が実験結果である．インデックスエントリの配置では並列的な処理を行うため，分割フレーム数の増大によるインデックスエントリ数の増大に対し，応答時間の増加曲線は緩やかである．また，どのシグネチャ長の場合も応答時間はほとんど変化していないことが分かる．

このように，分割フレーム数を大きくした場合にはメッセージ数は非常に大きくなる傾向にあるが，応答時間で見た場合には，フレーム分割を行わない場合と比べてもそれほど大きな差はない．

6.3 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数について検討する．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，そのときの検索時のメッセージ数と追加時のメッセージ数にそれぞれ

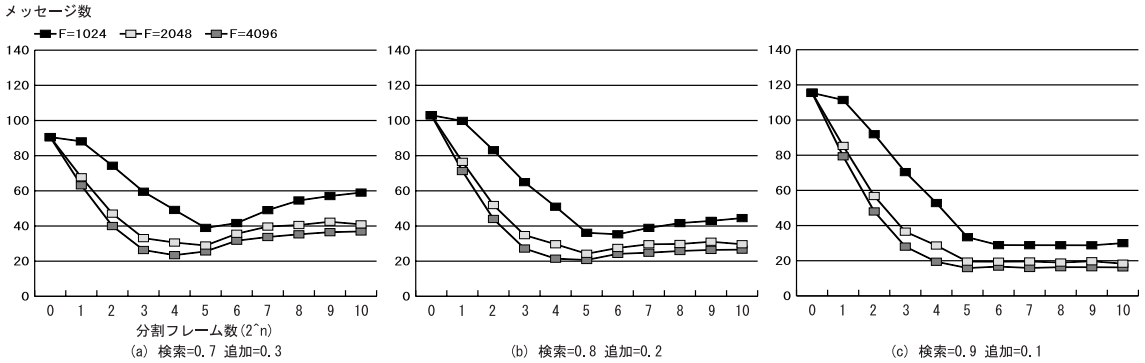

図 12 検索と追加が混在する場合のメッセージ数
Fig. 12 Number of messages for retrieval and index entry registration.

の生起確率をかけて合計したものを平均メッセージ数とする．

図 12 が実験結果である．一般的に検索処理の方が，追加処理と比べて多く発生すると考えられるため，p を 0.7，0.8，0.9 と変化させる．$p = 0.7$ の場合は，分割フレーム数が 2^4 のあたりでメッセージ数が最も小さくなる．また，$p = 0.8$ の場合は，分割フレーム数が 2^5 のあたりでメッセージ数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数はほぼ一定になっていることが分かる．このように，検索と追加の発生割合に応じて，最適の分割フレーム数が異なってくる．

6.4 オフラインノードが存在する場合の検索の精度

最後に，所定の割合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における，検索の精度について実験を行う．なお，Chord の枠組みではオフラインノードが存在する場合でもルーティング情報を動的に更新することができる．このため，ルーティング情報はつねに正しく利用できるものと仮定する．本実験では，問合せを実行した場合のフォールスドロップ確率を計算する．オフラインノードが存在する場合には，これらのオフラインノードに存在するフレームシグネチャとの照合ができないため，フォールスドロップ数が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 とし，正しく照合処理が行えないノードの割合を 0%から最大 50%まで 2.5%ずつ変化させる．

図 13 が実験結果である．オフラインノードの割合が大きくなるにつれて，フォールスドロップ確率も増加することを確認できる．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と考えられる．このように，オフラインノードが存在する状況下でも，分割フレーム数を大きくすることで一定水準の検索の精度を維持することができる．

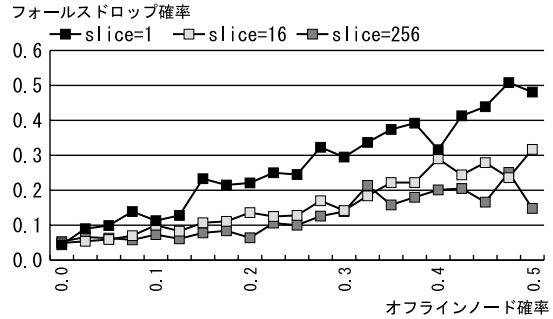

図 13 オフラインノードがあるときのフォールスドロップ確率
Fig. 13 False drop probability when some nodes are offline.

7. おわりに

本研究では，P2P 環境における分散配置されたシグネチャ情報を用いたオブジェクト検索方式を提案し，シミュレーション実験によりその基本的性質を考察した．一般に，分割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数，総データ転送量，応答時間を削減できることを示した．一方，分割フレーム数を増加させるとデータ追加の際のメッセージ数は増加するが，並列処理の導入により応答時間については大きな変化はないことを示した．検索と追加の発生割合を考慮した場合では，分割フレーム数を変化させることで処理効率を最適化できることを示した．このこと

は，発生割合に対して柔軟な対応ができることを意味している．さらに，オフラインノードが存在する場合の検索の精度についても実験を行い，特に分割フレーム数が大きい場合には，オフラインノードが存在する場合でも一定の検索精度を維持可能であることを示した．

今後の課題として，検索処理における並列処理方式の導入や，実際の計算機環境における検索時間，更新時間等の計測がある．また，各特徴量をキーとする転置ファイルを作成し，これを分散配置する方式も考えられる．この方式は本方式の特別な場合ととらえることが可能であり，それに関するより詳細な検討も興味ある検討課題である．

謝辞 本研究の一部は，日本学術振興会科学研究費萌芽研究（15650011），基盤研究（B）（15300027），若手研究（B）（14780316），文部科学省科学研究費特定領域研究（2）（15017207）による．

参 考 文 献

1) Aberer, K.: P-Grid: A Self-Organizing Access Structure for P2P Information Systems, *CoopIS 2001*, LNCS 2172, pp.179–194 (2001).
2) Faloutsos, C.: Signature files: Design and Performance Comparison of Some Signature Extraction Methods, *Proc. ACM SIGMOD 1985*, pp.63–82 (1985).
3) Freenet website. http://freenet.sourceforge.net/
4) Gnutella website. http://www.gnutella.com/
5) Hildrum, K., Kubiatowicz, J.D., Rao, S. and Zhao, B.Y.: Distributed Object Location in a Dynamic Network, *SPAA'02*, pp.41–52 (2002).
6) 石川佳治，北川博之，大保信夫：シグネチャファイルによる集合値検索のコスト評価，情報処理学会論文誌，Vol.36, No.2, pp.383–395 (1995).
7) Lin, Z.: Concurrent Frame Signature Files, *Distributed and Parallel Databases*, Vol.1, pp.231–249, Kluwer Academic Publishers (1993).
8) Lin, Z. and Faloutsos, C.: Frame-Sliced Signature Files, *IEEE TKDE*, Vol.4, No.3, pp.281–289 (1992).
9) Ratnasamy, S., Francis, P., Handley, M., Karp, R. and Shenker, S.: A Scalable Content-Addressable Network, *SIGCOMM'01*, pp.161–172 (2001).
10) Stoica, I., Morris, R., Karger, D., Kaashoek, M.F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *SIGCOMM'01*, pp.149–160 (2001).
11) Yang, B. and Garcia-Molina, H.: Improving Search in Peer-to-Peer Networks, *ICDCS'02* (2002).

(平成 15 年 3 月 25 日受付)
(平成 15 年 7 月 2 日採録)

（担当編集委員 仲尾 由雄）

松下 亮（学生会員）
2002 年群馬大学工学部情報工学科卒業．現在，筑波大学大学院理工学研究科修士課程在学中．P2P ネットワーク，情報検索等に興味を持つ．日本データベース学会学生会員．

北川 博之（正会員）
1978 年東京大学理学部物理学科卒業．1980 年同大学大学院理学系研究科修士課程修了．日本電気（株）勤務の後，1988 年筑波大学電子・情報工学系講師．同助教授を経て，現在，筑波大学電子・情報工学系教授．理学博士（東京大学）．異種情報源統合，文書データベース，WWW の高度利用等の研究に従事．著書「データベースシステム」（昭晃堂），「The Unnormalized Relational Data Model」（共著，Springer-Verlag）等．2003 年電子情報通信学会論文賞受賞．ACM SIGMOD 日本支部長．日本データベース学会，電子情報通信学会，日本ソフトウェア科学会，ACM，IEEE-CS 各会員．

石川 佳治（正会員）
1989 年筑波大学第三学群情報学類卒業．1994 年同大学大学院博士課程工学研究科単位取得退学．同年奈良先端科学技術大学院大学情報科学研究科助手．1999 年筑波大学電子・情報工学系講師．2003 年同助教授．博士（工学）（筑波大学）．2000 年度山下記念研究賞受賞．2003 年電子情報通信学会論文賞授賞．文書データベース，空間データベース，情報検索等に興味を持つ．ACM，IEEE-CS，電子情報通信学会，ACM SIGMOD 日本支部，日本データベース学会各会員．