SkipGraphを用いたデータの可用性向上のための複製配置手法の提案

奥　智照† 　上島　紳一†

† 関西大学大学院総合情報学研究科　〒 569–1095 大阪府高槻市霊山寺 2–1–1
E-mail: †{fb8m121,ueshima}@edu.kansai-u.ac.jp

あらまし　本稿では，構造型 P2P ネットワークのひとつである SkipGraph における，複製配置によるデータ可用性の向上について議論する．まず，順序付きキーの列に対して，資源を割り当て，キー順序を保持した SkipGraph を構成することで，範囲検索の可能な形でデータを自律分散的にデータ管理を行う方法を提案している．さらに，SkipGraph の隣人関係に従って，複製データの配置法について提案し，ネットワーク上からデータを消失を防ぐための自律分散的な修復法について議論する．提案手法では，データ管理層を用いることにより，SkipGraph のトポロジー管理とデータ管理を独立させている．最後に，提案手法について有効性を確認するため，数値シミュレーションによる評価を行い，その有用性を示す．

キーワード　SkipGraph, 複製配置, 自律分散, データ管理

Proposing Replica Allocation Scheme for Enhancement of Data Availability over SkipGraph

Tomoteru OKU† and Shinichi UESHIMA†

† Graduateschool of Infomatics, KansaiUniversity　Ryozenji 2-1-1,Takatsuki, 569–1095 Japan
E-mail: †{fb8m121,ueshima}@edu.kansai-u.ac.jp

Abstract　This paper studies a replica allocation scheme over a structured P2P network, SkipGraph. First, a method to construct SkipGraph network for a given key-ordered list of data, is provided by assigning nodes (=CPU) in logical key space to preserve key-order among data stored in subnetworks of higher layers. Also data management policies are discussed that enable range queries in terms of keys in autonomous P2P settings. Furthermore, a replica allocation scheme using SkipGraph topology is proposed, and discussion on repair strategies is also provided to avoid a complete loss of data from the network, due to the sudden disappear of nodes. In the approach, data management layer is separated from topology maintenance of SkipGraph, which enables to decrease complexity of replica management as well as network cost.

Key words　SkipGraph , Replica Allocation, autonomous, data management

1. はじめに

近年，クラウドコンピューティング [1] やサーバサイド P2P [2] に関する研究に注目が集まっている．これらの技術は, 利用者が, ネットワーク上の資源の位置や性能を明示的に指定することなく, 計算やデータ格納などのサービスを受けることのできる情報処理環境の実現技術として期待されている．特に, データ管理においては, 資源間での自律分散的なデータ管理機構の実現や，ネットワーク障害や資源自身の消失などに対して耐性を保持するデータ可用性の達成が重要な課題とされている．

本稿では，pure 型 P2P ネットワークの SkipGraph [3] における複製配置によるデータ可用性の向上について議論する．ここでは，まず，順序付きデータの列に対して，ノードを割り当て，データの範囲検索が可能な Skip Graph を構成する方法について述べ，ノードの管理領域について，分散ハッシュ表を用いた方式と比較しながら述べる．

次に，SkipGraph の隣人関係を利用した複製配置法について述べ，さらに，各ノードが他のノードと協調しながら，自律分散的にデータの管理や複製データの配置を行う方法について述べる．本方式は，SkipGraph のトポロジー変更による複製の再配置の負荷を軽減するため，データ管理を SkipGraph のノード間の隣人関係の管理とは分離して実行する手順になっており，複雑な SkipGraph のトポロジーに変更に対しても効率的に複製管理が行える特徴を持つ．提案手法によって，ノードが突然に脱退した場合でも，自律分散的にデータ再配置を行うアルゴリズムを用いることで，データ管理構造を維持することができ，

データの可用性を向上させることができる．

さらに，これらを踏まえた数値シミュレーションによる評価を行い，各データに対する複製データの総数が O($lonN$) と低く抑えられるにも関わらず，平均して 4 割強のノードがデータの引継ぎ処理を行わずにネットワークから脱退しても，データをネットワーク上に残せるをこと示す．

以後，2 章では提案手法の基本的なシステム構造を述べ，3 章で複製データの配置方法について述べる．さらに，4 章でデータの自律分散的な再配置法について提案し，5 章で数値シミュレーションによる評価を行う．

2. 基本的な考え方

本稿ではサーバ等の固定インフラを用いずに，オーバレイネットワークに参加しているノード（通信可能な機器）を利用し，ネットワーク上でデータを管理することを目標としている．ここでは SkipGraph をはじめとする構造型 P2P ネットワークを用いたデータ管理の利点について述べ，提案データ管理の構造と，SkipGraph [3] について簡単に説明する．

2.1 構造型 P2P ネットワークと複製配置

P2P ネットワークに参加しているノードが，管理しているデータの引継ぎを行わずに脱退した場合でも，データの消失を防ぐためには，他の複数のノードに複製データを配置しておくことが有効である．これまで，非構造型，構造型のネットワークの双方に対して様々な先行研究がある．

本稿では，SkipGraph を用いて，範囲検索を可能にしながら，データの可用性を向上させるために複製配置を行う方法について述べる．SkipGraph は，他の構造型 P2P ネットワークと同様に，データの位置を構造に従って特定できる利点を持つ．さらに，この特徴を生かして，SkipGraph の隣人関係に従って複製データを配置し，複製データの探索や管理を容易にする手順について議論する．反対に，構造型 P2P ネットワーク上で，複製データを無作為に配置する方法は，複製データの探索に手間がかかり，非構造型 P2P ネットワーク上でフラッディング等により探索する手法と同じになり，得策でない．

また，SkipGraph のような構造型 P2P ネットワーク上でデータを管理した場合には，ノード間の接続変更に対してデータの再配置が必要になる．そのため 4 章で自律分散的なデータの再配置手法について述べる．

2.2 データ管理層とネットワーク層

図 1 にネットワークとデータ管理に関する階層図を示し，各層の動作を説明する．

［IP ネットワーク層］ IP ネットワーク層ではインターネットプロトコル (IP) を用いたデータ通信を行う．

［構造型 P2P ネットワーク層（SkipGraph 層）］ 構造型 P2P ネットワーク層では隣人ノードとのみ通信が可能な構造型 P2P ネットワークが定義されており，最下層の IP ネットワーク層を用いて，ネットワークを介したデータの送受信を行う．特に，本稿ではこの層に SkipGraph を用いる．また，データ管理は後述するデータ管理層が受け持つため，この層上ではデータの管理を行わない．

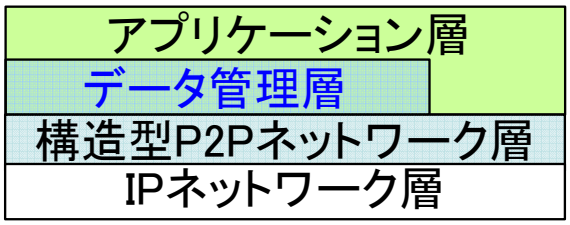

図 1 階 層 構 造
Fig. 1 MultiLayer

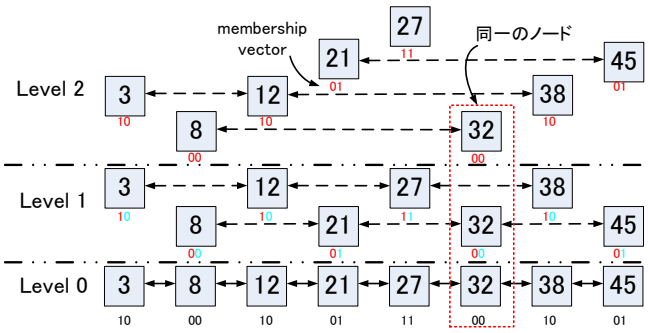

図 2 SkipGraph の例
Fig. 2 An example of SkipGraph

［アプリケーション層］ ここでは，クラウドコンピューティングなどのアプリケーションが動作する．アプリケーションは SkipGraph を通してデータ送信を行う．

［データ管理層］ データ管理層は，SkipGraph の隣人関係を利用しながら，上位層のアプリケーションデータを管理し，可用性向上のためデータ複製の配置などを行う．

2.3 SkipGraph

SkipGraph [3] は，図 2 に示すような接続関係持つ構造型 P2P ネットワークの一種である．図 2 では正方形がノードを表し，その中の数字は順序付け可能なキーである．また，ノードはキーの順に整列しており，ノード下の数字は membership vector と呼ばれるノードに与えられる 2 進数の値である．各ノードは membership vector が先頭から $i(i = 0,1,2\cdots)$ 文字等しいものと部分ネットワークを形成する．その各部分ネットワークにおいてキーの順に整列し，キー値が自身と前後に最も近いものと双方向の P2P 接続を確立する．特に，membership vector が頭から i 桁等しいものを Level i の接続と呼ぶ．ノードは membership vector で識別するが，説明のために Nn は識別子 n のノードを指し，$Nn.Key$ は識別子 n のノードのキー (以後ノードキー) をさすものとする．また，SkipGraph のルーティング手法および生成手法などは文献 [3] に依るものとし，ノード間通信は全て SkipGraph を用いるものとする．SkipGraph は常に正常な接続関係を維持でき，自律分散的に動作するものとしている．また，冗長化のために各部分ネットワークの最小と最大キーを保持するノード同士に対して接続関係を持たせリング構造にする．

3. 提案データ管理方式

データには以下の情報が付与されるものと仮定する．

（1） 順序付けが可能なデータのキー (以後データキー)

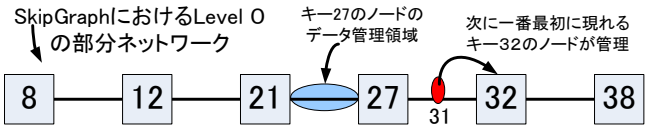

図 3　データ管理領域

Fig. 3　Data Management Area

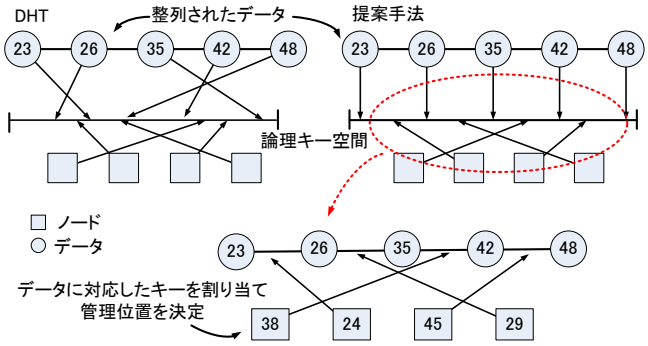

図 4　データとノードの論理キー空間への配置

Fig. 4　Allocation of Node and Data for Logical Key Space

（2）　データ複製を依頼してきたノードキー

(1) はデータを管理するノードを決定するために必要であり，(2) は 4. 章で説明するデータ管理構造維持のために用いる．

3.1　データの管理法

データの管理は SkipGraph の Level 0 の隣人関係を利用して，論理キー空間上での隣接関係を基準に行う．すなわち，各ノードのノードキー，$N_1.Key$，$N_2.Key$，…，$N_n.Key$ とし，$N_1.Key < N_2.Key < \cdots < N_n.Key$ の順に整列されている．このとき，ノード N_i（$1 < i \leqq n$）が管理するデータ領域のキーを X とすると，次式を満たす領域のデータを管理する．

$$N_{i-1}.Key < X \leqq N_i.Key$$

図 3 はその様子を示した図である．同図では，ノードキー 27 のノードはデータキー 21 から 27 の間にあるデータを管理し，データキー 31 のデータはノードキー 32 のノードに管理される．

Note:　以後，データキーによりデータを管理するノードが決定するデータを**管理データ**と呼ぶ．また，データ管理層は SkipGraph(構造型 P2P ネットワーク層) とは別の層であるが，本稿では簡潔に説明するためにデータとノードを**擬似的**に同一キー空間に配置しているとして議論を進める．

3.2　ノードとデータにおける論理キー空間への配置

論理キー空間へのデータ配置は，データに割り当てられる整列可能なデータキーを用い，ノードはそのデータキーに対応するノードキーを割り当てることにより論理キー空間へ配置する．

図 4 にＤＨＴ (分散ハッシュテーブル) と提案手法を用いた場合の論理空間へのデータ配置法を示す．ＤＨＴのようにハッシュ関数を用いてキー空間上にデータを配置した場合，順序づけされていたデータの順序が崩れてしまい，その順序を用いた範囲検索ができないという問題がある (図 4 左上)．そのため，本稿ではデータに対してはその順序を崩さずにそのまま論理キー空間にその順序の規則どおりに割り当て，ノードにはデータキーに対応したノードキーを割り当てる (図 4 右上)．

これによりデータは DHT と異なりハッシュ変換を行わない

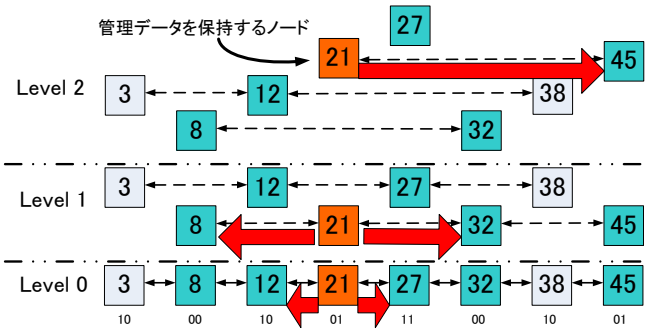

図 5　データの複製配置法

Fig. 5　Data Replication Scheme

ため，データキーの順序がランダムにならず範囲検索が可能な構造を維持しながら論理キー空間への配置が可能になる．ただし，ハッシュ関数を用いた論理キー空間への配置のように，データ配置が論理キー空間上で均一にならないため，ここでは**データの全体数が既知**であると仮定し，ノードはデータの管理数が均一になるように配置できるものとする．

図 4 下にノードの論理キー空間への配置例を示す．整列可能なデータキー (図では数値) の順に並んだ論理キー空間上にそのデータキーに相当するノードキーを割り当てる．同図では，データキーと数値キーをノードキーとして，論理キー空間上に配置し，管理するデータ位置を決定している．つまり，整列が可能なデータキーに対応したノードキーをノードに割り当てることにより，管理データを保持するノードを決定する．

この手法は順序付けされたデータのキー空間上にノードを埋め込むことにより，管理データを保持するノードを決定する方法である．また，本稿における SkipGraph はデータの転送やデータの複製配置基準などの索引構造を提供している．これは 2.2 節で述べたとおり，データ管理を行う層を構造型 P2P ネットワークから分離して考えているためである．

3.3　複製データの配置法

P2P ネットワークに参加する全ノードに複製を配置すればデータの消失を防げるが，それは現実的ではない．また，非構造型 P2P ネットワーク上でアクセス頻度などを考慮し複製データを配置する手法にはパス複製法 [4] やオーナー複製法 [5] が提案されている．

しかし，構造型 P2P ネットワークを用いて複製データを管理する利点は複製データをどこに配置するかを一意に決定可能な点である．そのため，従来の複製配置手法ではなく，構造型 P2P ネットワークの接続関係を利用して複製配置を行う．そこで，本稿では SkipGraph の隣人ノードに対して複製データを配置することを提案する．

図 5 は提案する複製データを配置するノードを示した図である．図 5 のノードキー 21 のノードは矢印に示すノードキー 8, 12, 27, 32, 45 の隣人ノードに複製データの保持を依頼する．

SkipGraph の構造には複数の隣人ノードが脱退しても構造を修復できる特徴がある．そのため，複製データは管理データと同一のものであると考えた場合，複製データを SkipGraph の全隣人に格納しておくことにより，隣人ノードがネットワー

```
Algorithm 1 CheckDeleteofReplicationData()
 1: boolean changeNegibor = false;
 2: for i = 0 to ReplicationDataList.end();
 3:     if (!CheckChangeNeigbor (ReplicationDataList[ i ] , NNList))  then
 4:         if (CheckNeigbor hasData(ReplicationDataList[ i ].Key) )
            then  /* true processes */
 5:             if CompleteRelocation(ReplicationDataList[ i ])  then
 6:                 CLEAR ReplicationDataList[ i ];
 7:             else
 8:                 changeNegibor = true;
 9:             end if
10:         else /* pseudo processes */
11:             send ReplicationDataList[ i ]
                    to getNode(ReplicationDataList[ i ].Key);
12:             changeNegibor = true;
13:         end if
14:     end if
15: end for
16: if (changeNegibor)  then
17:     CheckDeleteofReplicationData();
18: end if
```

図 6　複製データの削除確認アルゴリズム
Fig. 6　An Algorithm of Delete Replication Data

ク上に生存する限り，自身が管理するデータの消失が防ぐことが期待できる．また，SkipGraph における隣人関係の変更時のみ，複製データの再配置が必要になる．そのため，複製データの再配置処理は隣人関係変更時のみに限定できる．

4. 複製データの管理アルゴリズム

ここでは，提案する複製データ配置の一貫性を実現するための，自律分散的な構造維持手法について述べる．

4.1 データの更新, 追加削除

[**データの更新および削除**] データの更新，削除などの変更は管理データだけでなく，全ての複製データに対しても反映する必要がある．そのため，はじめに管理データを変更し，その後複製データに対しても変更を反映する．提案手法では，管理データを保持するノードの隣人ノードに複製データを配置しているため，複製データの更新および削除は管理データを保持するノードが，SkipGaph の隣人ノードに対して，データの更新を依頼すればよい．

[**データの新規追加**] 提案するデータ管理システムに対して，新たなデータを追加する方法は以下の手順で行う．

まず，データキーを基に管理データを保持すべきノードを決定し，そのノードが管理データを保持する．その後，管理データを保持するノードが SkipGraph の隣人関係を利用し，複製データを配置する．

4.2 ノードの参加脱退に伴う複製データの再配置

提案手法は隣人ノードに対して複製データを配置しているため，隣人関係の変更により複製データを再配置する必要がある．ここでは，隣人関係の変更に伴う自律分散的な管理データの引継ぎ方法，ならびに複製データの再配置手法を提案する．

また，本節では管理データを保持するノードが管理データの委譲等を行わずに P2P ネットワークから脱退した場合にも，複製データから管理データを復元する方法を提案する．

4.2.1 複製データの削除

本節では，隣人関係の変更に伴う複製データの削除に方法ついて述べる．複製データを保持するノードは，SkipGraph の隣人に管理データを持つノードが存在する．さらに，複製データのノードキーから，SkipGraph の隣人関係が変更した場合に，管理すべきでない複製データかを判断できる．しかし，隣人関係が変更したからといって，複製データをすぐに削除したのでは P2P ネットワーク上でデータの存在保証のために複製データを配置した意味がない．

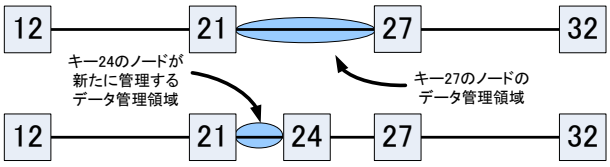

図 7　新規参加ノードに対する管理領域の引継ぎ
Fig. 7　Take over Management Area to Join Node

そこで，図 6 に示す手順により，隣人ノードから依頼された複製データのみを管理し，データ消失を防ぎながら隣人ノードのものでない複製データの削除方法を提案する．同図において複製データは *ReplicationDataList* に格納されており，管理している全ての複製データに対して削除確認処理を行う．図中で 3 行目は複製データに付与されているノードキーをもとに，複製データの保持を依頼してきたノードが SkipGraph の隣人に存在するかを確認している．もし，隣人ノードに複製データの保持を依頼してきたノードが存在しなければ，その複製データを削除する必要がある．その削除処理を 4 行目から 13 行目にデータ消失を防ぐ複製データの削除処理を示す．

まず，複製データのデータキーを基に新たに管理データとなるべきノードに問い合わせを行う．隣人関係の変更に伴い新たな管理ノードが，複製データの再配置が完了していればその複製データを削除する (図 6，5 行目)．再配置が完了していなければ，複製データの削除を保留する．(図 6，8 行目) 新たにその複製データの管理データを保持すべきノードがそのデータを保持していなかった場合，複製データから管理データの復元を行い，複製データの削除を保留する (図 6，10 から 12 行目)．最後に，削除を保留している複製データのがあれば再度削除確認処理を実行する．(図 6，16 から 18 行目)．

4.2.2 隣人関係変更時のデータ管理

ここでは，隣人ノードが変更された場合の自身が持つ管理データの複製処理と，複製データの処理について述べる．

[**複製データの配置**] 管理データを保持していれば，隣人関係変更に伴い，管理データの複製を配置しなければならない．そのため，管理データを保持するノードは接続関係の変更時に変更された隣人ノードに対して複製データの保持を依頼する．

[**複製データの削除**] 4.2.1 節に述べた処理を実行することにより，隣人でなくなった複製データを削除する．

4.2.3 ノードの新規参加

新規参加ノードは，3 章で述べたようにデータキーに基づいた空間のデータを管理することになる．そのため，SkipGraph の参加ノードから自ノードが管理すべきデータを引き継ぐ必要がある．SkipGraph 上に $N_{n-1}.Key < N_n.Key$ を満たすノードが存在としたとき，$N_{n-1}.Key < N_{new}.Key < N_n.Key$ をみ

たすノード N_{new} がネットワークに参加する．このとき，ノード N_n は次式の領域データ管理を委譲することになる．

$$N_{n-1}.Key < X \leqq N_{new}.Key$$

新規参加ノードに対するデータ引継ぎ領域の例を図 7 に示す．図では，ノードキー 27 のノードが管理していたデータキー領域に対して，新たに参加したノードキー 24 のノードが引き継ぐ領域を示している．ノードキー 24 を持つノードは SkipGraph に新規参加したとき，その隣人関係からデータキー 21 から 24 の領域を管理することになる．同時にノードキー 27 を持つノードは隣人関係の変更により，データキー 24 から 27 の領域を新たに管理することがわかる．そのため，ノードキー 27 を持つノードは 21 から 24 の領域に位置する管理データを新規参加ノードであるノードキー 24 のノードに引き継ぐ．その後，ノードキー 27 のノードは管理領域変更を隣人に伝える．

管理データの引継ぎ後は，新規参加ノードが隣人ノードに対して複製データの保持を依頼する．こうして，新規参加ノードにおいて自律分散的にデータの再配置を行う．

また，管理データの受け渡しにおいて影響するノードは 1 つであり，複製データにおいても影響を及ぼすノードは管理データを保持するノードの隣人ノードに限られる．そのため，SkipGraph の隣人ノードの個数は O($logN$) であることから，再配置の影響するノード数は O($logN$) であると期待できる．

4.2.4 ノードの脱退

ノードの脱退は管理データの引継ぎ等を行わなずに P2P ネットワークから突然脱退することを想定する．

[Level 0 の隣人ノードの脱退] Level0 の隣人ノードの脱退では，ノードが管理するデータ領域の変更が行われる．そのため，参加ノードで脱退したノードが管理していた領域を引き継ぐことになる．$N_{n-1}.Key < N_n.Key < N_{n+1}.Key$ を満たすノードが存在するとき，N_n が脱退した場合，ノード N_{n+1} が N_n のデータ管理領域を引き継ぐことになる．つまり，SkipGraph に参加しているノードの中で脱退ノードよりノードキーが次に大きい隣人ノードがその管理領域を引き継ぐ．

複製データの中に新たな管理領域のデータを保持していれば，その複製データを管理ノードとし，自身の隣人ノードに対して複製の再配置を行う．保持していなければ，4.2.1 節で述べた削除可能かの問い合わせ時にデータを送信してもらい，ノード脱退後の正常な複製配置を得る．

[Level 0 以上の隣人ノードの脱退] Level 0 以上の隣人ノードの脱退では，新たにデータを管理する処理は行われない．そのため，4.2.2 節の処理を実行すればよい．

Note: ノードの脱退に伴い，脱退ノードが管理していた領域を新たに管理するノードは脱退したノードの複製データを保持している．これは，通常のノード脱退における引き継ぎ領域は SkipGraph の Level 0 の隣人が管理するためである．また，ノードの脱退においてデータが修復ができない状態には以下の 2 つの状況が考えられる．

（1） 管理データおよび複製データを保持したノードがデータの再配置を行う前にすべての脱退した場合．

（2） SkipGraph の隣人ノード全て脱退し，複製データの再配置を行うための通信できなくなった場合．

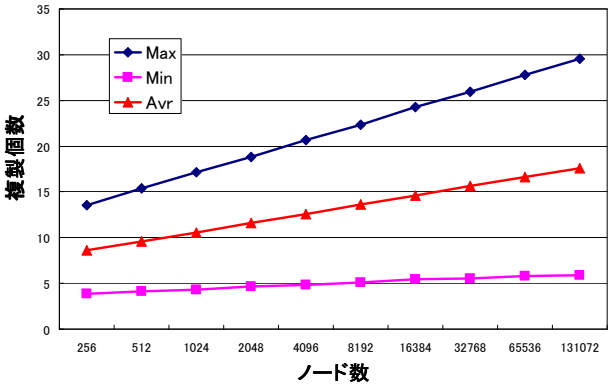

図 8 複製データの保持個数

Fig. 8 Number of Data Replica

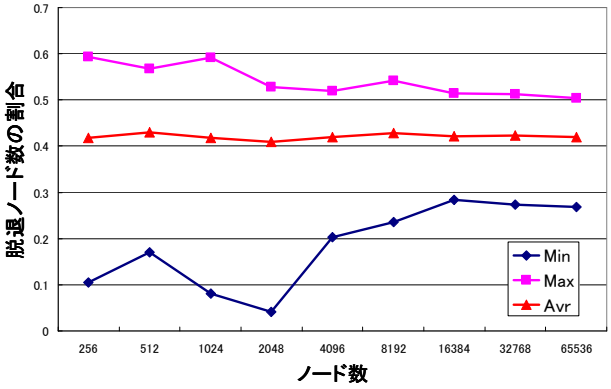

図 9 データ消失時のノード脱退数の割合

Fig. 9 Ratio of Number of Nodes When it can't Repair data

5. シミュレーションによる評価

本章では，提案データ管理構造の有効性を検証するため複製データの保持個数ならびに，ノードをランダムに脱退させた場合，データがネットワーク上から喪失時の残存ノード数を計測する．また，ノード総数を N としノードおよびデータを [0,1] の領域に一様分布させる．

5.1 複製データの保持個数

図 8 に各ノードが一つの管理データを保持すると仮定した場合の，複製データの保持数の平均と最大および最小の平均値を示す．複製データは各ノードの隣人ノードに格納している．そのため，複製データ数は SkipGraph の隣人ノード数と等しくなる．つまり，複製データの保持個数は SkipGraph の隣人数と同様に O ($logN$) に抑えられる．

5.2 ネットワーク上からの完全なデータ消失

図 9 はノードをランダムに選択してネットワークから脱退させた場合，どの程度のノードが複製データの再配置前に脱退すれば，データ (管理データと複製データ) がネットワーク上から喪失するかを測定したものである．

同図に示すとおり，複製データの再配置が行われなくても，平均して 4 割近いノードが一度にネットワークから脱退しない限り，ネットワーク上に少なくとも一つのデータ (管理データまたは複製データ) が存在することがわかる．そのため，提案

する複製配置手法は高いデータの修復可能性が期待できる．

6. 関連研究

Cohen ら [4] は非構造型 P2P ネットワーク上における理想的な複製配置個数について議論している．その中で，問い合わせ率に対する複製数の比を平方根配置に近づけることでネットワークの負荷が軽減することを解析的に証明している．

非構造型 P2P ネットワーク上では複製データがどこに存在するかわからない欠点がある．そのため，複製データの更新情報が全ての複製データ反映することは困難である．渡辺ら [6] はこの欠点を克服するため，データごとに更新伝播木を生成することにより，非構造型 P2P ネットワーク上でも複製データの更新を効率的に行う手法を提案している．

さらに，DHT を用いた分散ストレージとしては Tapestry [7] を用いて実装した OceanStore [8] や，Pastry [9] を用いた PAST [10] などがある．

小西ら [11] は SkipGraph のキー (データ) とそれを保持するノードが１対１に対応し，各ノードには１つのキーしか保持できないという欠点を克服するため，１つのノードに複数キーを保持可能な方法を提案している．

7. おわりに

本稿では，データ管理層を用いてデータ管理を SkipGraph から分離する手法と，そのデータ管理アルゴリズムについて述べた．また，データを配置する際にデータのキー順を崩さず論理キー空間に配置することで，データの範囲検索が可能である．さらに，数値シミュレーションによりデータを保持している複数のノードが SkipGraph から突然脱退しても，提案する複製データの配置構造を修復し，維持できることを示した．

今後の課題としては，修復手順の通信回数などの定量的な評価や，アクセス頻度や検索効率などを考慮した複製個数の柔軟な変更手法やデータ配置の負荷分散手法の提案．さらに，実環境への適応を考慮した同期処理などが必要である．

文　献

[1] “IBM Blue Cloud”, http://www-03.ibm.com/press/us-/en/pressrelease/22613.wss, (2007)

[2] DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G. Lakshman, A, Pilchin, A, Sivasubramanian, S., Vosshall, P., Vogels, W., “Dynamo: amazon’s highly available key-value store”, SOSP ’07: Proc. 21st ACM SIGOPS Symp. on Operating systems principles, pp.205–220 (2007)

[3] Aspnes, J., Shah, G., “SkipGraph”, SODA ’03: Proc. 14th annual ACM-SIAM Symp. on Discrete algorithms, pp.384–393 (2003)

[4] Cohen, E,. Shenker, S., “Replication Strategies in Unstructured Peer-to-Peer Networks”, Proc. ACM SIGCOMM’02, pp.177–190 (2002)

[5] Lv, Q., Cao, P., Cohen, E., Li. K., Shenker, S., “Search and Replication in Unstructured Peer-to-Peer Networks”, Proc. ICS’02, pp.84–95 (2002)

[6] 渡辺俊貴, 神崎映光, 原隆浩, 西尾章治郎, “P2P ネットワークにおけるデータアクセス頻度を考慮した更新伝播法”, 情報処理学会論文誌 Vol.49 No.6, pp.1819–1832 (2008)

[7] Ben, Y.Zhao ., John D.Kubiatowicz., Anthony D.Joseph., “ Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and, University of California at Berkeley”, University of California at Berkeley, (2001)

[8] John, K., David B., Yan, C., Steven, C., Patrick, E., Dennis, Geels., Ramakrishna, Gummadi., Sean, R., Hakim, W., Chris, W., Ben, Z., “OceanStore: an architecture for global-scale persistent storage”, Proc. 9th international conference on Architectural support for programming languages and operating systems, pp.190–201, (2000)

[9] Antony I. T. Rowstron , Peter Druschel., “ Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems”, Proc. IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, pp.329–350 (2001)

[10] Peter, D., Antony, R., “PAST: A Large-Scale, Persistent Peer-to-Peer Storage Utility” Proc. 8th Workshop on Hot Topics in Operating Systems, pp.75 (2001)

[11] 小西佑治, 吉田幹, 竹内　亨, 寺西裕一, 春本要, 下條真司, “単一ノードに複数キーを保持可能とする Skip Graph の拡張”, 情報処理学会論文誌 Vol.49 No.9 pp.3223–3233 (2008)