社団法人　情報処理学会　研究報告
IPSJ SIG Technical Report

2006－AL－105（6）
2006／3／17

分散データ構造スキップグラフの探索頻度偏りを考慮した拡張について

原口 高裕，泉 泰介，角川 裕次，増澤 利光
大阪大学大学院 情報科学研究科
t-haragt@ist.osaka-u.ac.jp

抄録 — スキップグラフは資源の検索・挿入・削除を $O(\log n)$ (n は資源数) で実行可能な分散データ構造であり，故障耐性，負荷分散特性に優れている．スキップグラフでは全資源を対等に扱っているため，各資源に対するアクセス頻度に偏りのあるシステムにおける検索時間は最適ではない．そこで本稿ではアクセス頻度の偏りに対応するため，各資源に対して検索頻度に応じた重みを付加し，重みの大きい資源に対しては高速にアクセス可能となるようにスキップグラフを拡張する．本稿では，ある資源 i の重みを $w(i)$，全資源の重み総和を W とした際に，資源 i の検索時間が $O(\log(W/w(i)))$ となるような重み付け手法を提案する．また，資源の検索頻度が既知の場合に重みを割り当てる際，検索時間が最小となる重み付け方法を示す．この方法は理論的に最適であるが，メンテナンスコストの観点からは実用的ではない．そこで，メンテナンスコストを考慮した重み付け手法もあわせて提案し，その有効性をシミュレーションにより評価する．

An Extension of Skip Graph Considering Nonuniform Search Frequency

Takahiro Haraguchi, and Taisuke Izumi, and Yuji Kakugawa, and Toshimitsu Masuzawa
Graduate School of Information Science and Technology, Osaka University

Abstract — Skip Graph is a distributed data structure superior to fault tolerance and load balancing. It achieves $O(\log n)$ time complexity for search, insert, delete of resources, where n is the number of resources in the skip graph. While the search time is asymptotically optimal, it can be optimized when the search frequency of each resource is uniform. In this paper, we propose an extension of Skip Graph considering nonuniform access frequency. In our scheme, each resource has a weight, which implicitly implies search frequency of the resource. The search time of our scheme is $O(\log(W/w(i)))$, where W is the sum of the weight of all resources, and $w(i)$ is the weight of the target resource. We also prove that the total seach cost is minimized when every resource has the same ratio of its weight to its search frequency. However, this optimal weight assignment is impractical because of large maintenance cost. Thus, we propose two other practical methods of weight assignment, and show that those methods considerably reduce the total search cost by simulation.

1　はじめに

Peer-to-Peer(P2P) システムとは，中央サーバを使用しないような分散システムの一種である．P2P システムに参加する計算機はピアと呼ばれ，サーバを介さず直接接続されている．P2P システムは実ネットワーク上に仮想的に構成されたオーバーレイネットワークであり，ピア間の通信は論理的なリンクを通じて行われる．P2P システムでは参加するピア同士がサーバを介さず直接通信によって処理を行うため，負荷や処理の分散化，システムのスケーラビリティ，故障耐性など，多くの好ましい特性を有しており，近年様々な分野への応用が期待されている．応用例の中で代表的なサービスとしてはファイル共有システムが挙げられる．これは各計算機が自身の保有するファイルの一部を提供することで，全体として大量のファイルを共有するシステムである．

P2P システムにおいて資源（例：ファイル）は膨大な数のピアに分散して配置されているため，ファイル共有などのアプリケーションを動作させる場合，目的となる資源を持つピアを探し出す事が必要となる．この問題は探索問題と呼ばれ，P2P システムにおける基本問題の一つとして重要である．

探索問題の解決法として，初期の P2P システムでは Flooding 等の手法がよく用いられている．しかし，Flooding に基づく方法では，ピア数の増加に伴い大量のネットワーク資源を浪費してしまうため，スケーラビリティに問題があった．そこで近年，資源の検索キーからその資源を保有するピアを一意に特定し，特定された資源保有ピアに対して効率的なルーティングを行うことで高速かつスケーラブルな探索を実現する，分散データ構造に関する研究が盛んに行われている．最も代表的な例は，分散ハッシュテーブル (DHT)[8, 6, 9, 11] である．DHT では，ハッシュ関数を利用する事で資源とそれを保持するピアを対応付ける．また，DHT ではピアの ID からピアを探索しやすいようなネットワーク構造を論理的に構成しているため，探索時間が非常に高速であり，スケーラビリティの問題も大幅に改善されている．例えば DHT の一種である Chord[6] では資源の検索にかかる時間は $O(\log n)$ である (n はピア数)．DHT の一種として近年スキップグラフ [2, 4] という分散データ構造が考案されている．これはスキップリスト [7] というデータ構造を基として分散環境に対応するための拡張を施したものである．スキップグラフは検索時間 $O(\log n)$ を達成しており，更に故障耐性・負荷分散などの点において他の DHT よりも優れている．

スキップグラフでは全ての資源を平等に扱っており，どの資源を検索する場合にも期待値的には同じ検索時間を要する．しかし，実際のシステムでは各資源のアクセス頻度に大きな差がある場合が多く，そのような場合はアクセス頻度の低い資源の検索コストを犠牲にしても，アクセス頻度の高い資源をより高速に検索可能であるほうがシステム負荷，総合的な応答速度などの面で有利と言える．

そこで本研究では，資源に重みの概念を導入し，重み

の大きい資源ほど高速に検索可能とするようにスキップグラフを拡張する．スキップグラフの元となったスキップリストに対しては，既に重み付けの拡張を行ったバイアス付きスキップリスト [3] が考案されており，本研究における重み付けの手法はこの結果に基づいている．提案する手法のアイデアは，各資源が，与えられた重みに応じた個数の複製を持つことである．この重み付きスキップグラフ上における資源の探索に要する時間は $O(\log W/w(i))$ となる．ここで $w(i)$ は資源 i の重みであり，W は全資源の重み総和である．本研究ではさらに，検索頻度が与えられたとき，システム全体で検索に要する総メッセージコストが最小となるような重み付けの方法を提案する．この重み付けの手法は理論的には最適なコストを達成するが，実際にそのような重み付けを行ったとき，多くの場合において複製の数が莫大になるため，記憶容量，メンテナンスコストの面で現実的ではない．そのため，探索時間を削減しつつ複製の数を制限する手法を 2 種類提案し，その有効性をシミュレーションにより検証する．

本稿の構成は次のとおりである．2 節では，本稿を通して使用するシステムモデルについての定義を行う．3 節では，本研究の基となったスキップグラフの紹介を行い，4 節では，スキップグラフに対し重み付けを行う方法と検索時間の評価，検索に要する時間を最小化する方法について述べる．5 節では，複製の数を制限する方法を 2 種類提案し，性能をシミュレーションによって比較・評価する．最後に 6 節で本稿の結果をまとめる．

2 モデル

複数のプロセスがメッセージ交換により通信を行う動的分散システムを考える．システム内に存在するプロセスの集合はプロセスの参加・離脱によって動的に変化する．本稿ではプロセスの参加・離脱について触れないが，スキップグラフ [2] と同様の方法で対応することが可能である．

動的分散システムは，システム内に存在するプロセスの集合と，それらを相互接続する通信リンクから構成される．各プロセスは通信リンクを介して相互にメッセージ交換を行う．通信リンクは実ネットワーク上に論理的に構成されたものであるため，システムに参加している任意の 2 プロセス間に構成可能である．通信は信頼できるものとし，メッセージの消失・複製・改変は起こらないものとする．更にメッセージの通信遅延には上界が存在し，各プロセスはその上界を知っていると仮定する．

また本稿では計算時間に関して次の仮定をおく．

- 通信リンクを通したメッセージの配送には高々1 単位時間を要する．
- プロセスの内部処理 (値の計算・評価等) に要する時間は無視できる．

3 既存研究：スキップグラフ

本節では本研究の基となる分散データ構造スキップグラフ [2] について説明する．

3.1 構造

スキップグラフは，複数の双方向リストで構成される分散データ構造である．リストはノードと，ノード間の双方向通信リンクで構成される．ノードは検索キーを持っており，検索キーは順序関係を持つものとする．同じリスト内においてノードは検索キーに関して昇順に整列している．また，各ノードにはメンバシップベクタと呼ばれる無限長のランダムな文字列が割り当てられている．メンバシップベクタの各文字は文字集合 Σ に含まれる文字のいずれかであるとし，$k = |\Sigma|$ とおく．検索キーが x であるようなノードを以降ノード x と呼び，ノード x のメンバシップベクタを $m(x)$ とする．

スキップグラフ中の各リストはレベルを持つ．レベル 0 のリストには全ノード (合計 N ノード) が出現する．レベル l では，全ノードはメンバシップベクタの l 文字目までの接頭部により分類され，分類されたグループがそれぞれ別の双方向リストを構成する．例えば $\Sigma = \{0,1\}$ とした場合，レベル 1 においては全ノードはメンバシップベクタの先頭部が 0 のノードと 1 のノードに区別され，それぞれが独立なリストを構成する．同様に，レベル 2 はベクタの接頭部が 00,01,10,11 の 4 グループに分類され，それぞれが独立にリストを構成する．この分類は，リストに出現するノード数が 1 となるレベルまで繰り返される．

メンバシップベクタは一様ランダムな文字列であるため，各ノードは各レベルにおいてバランス良く振り分けられることになる．図 1 に $\Sigma = \{0,1\}$，$N = 6$ のスキップグラフの例を示す．

3.2 P2P 環境での実装

P2P ネットワーク上にスキップグラフを実装した場合，スキップグラフの各ノードは 1 つの情報資源に対応する．各ピアには複数のノードが割り当てられており，スキップグラフ上の資源の接続関係は，その資源を管理するピア同士の接続関係に対応する．すなわち，スキップグラフ上でリンクを辿ることは，移動先のノードの検索キーを管理するピアに対してクエリを転送することに対応する．ピアに対してキーを割り当てる方法については様々な方法が存在するが [1]，単純なものとして次の 2 つがある．

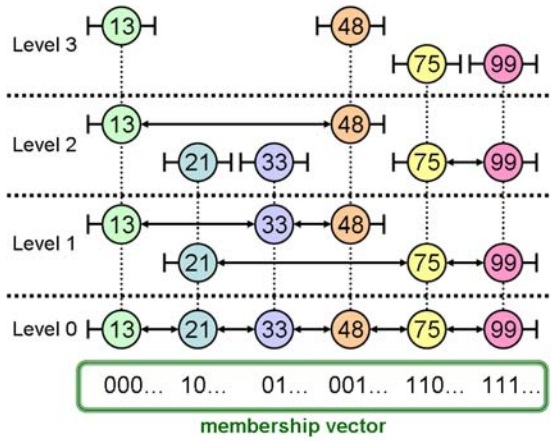

図 1: $\Sigma = \{0, 1\}, N = 6$ のスキップグラフ

- ピアがシステムに参加する際に保持している資源については全てそのピアにより管理される．
- 各ピアにはあらかじめ決められたハッシュ関数 H の出力の一定範囲が担当として与えられ，各資源は自検索キーをハッシュ関数 H にかけた出力を担当しているピアによって管理される．

本稿では簡単のため，各資源は必ず一つのピアにより管理されているとするが，どの資源がどのピアに対応しているかということについては特に考えない．なお，以降ノード v に対応するピアを単にピア v と呼び，ノード v に対応するピアの動作を単にノード v の動作と呼ぶことがある．

3.3 検索方法

スキップグラフ上で検索を行う際のアルゴリズムについて説明する．検索クエリを生起したノードを s，検索対象ノードを d とする．

検索アルゴリズムの擬似コードを図 2 に示す．擬似コード中で新たに用いる表記の意味は次のとおりである．

- <searchOp, $startNode$, $searchKey$, $level$> ： 検索クエリを表す．各項の意味は以下．
  - searchOp : メッセージ種別．この場合はメッセージが検索クエリであることを表している．
  - $startNode$: 検索クエリを生起したノード．このノードに検索結果が返される．
  - $searchKey$: 検索対象ノードの検索キー．
  - $level$: 現在どのレベルのリストを辿っているかを表す．このレベル以下の隣接ノードへクエリは転送される．

	ノード n が検索クエリを受け取った場合の動作アルゴリズム
(1)	upon receiving <searchOp, $startNode$, $searchKey$, $level$>:
(2)	**if** $n.key = searchKey$ **then**
(3)	**send** <searchOp, n> to $startNode$
(4)	**if** $n.key < searchKey$ **then**
(5)	**while** $level \geq 0$ **do**
(6)	**if** $(nR_{level}).key < searchKey$ **then**
(7)	**send** <searchOp, $startNode$, $searchKey$, $level$> to nR_{level}
(8)	**break**
(9)	**else** $level \leftarrow level - 1$
(10)	**else**
(11)	**while** $level \geq 0$ **do**
(12)	**if** $(nL_{level}).key > searchKey$ **then**
(13)	**send** <searchOp, $startNode$, $searchKey$, $level$> to nL_{level}
(14)	**break**
(15)	**else** $level \leftarrow level - 1$
(16)	**if** $level < 0$ **then**
(17)	**send** <searchOp, n> to $startNode$

図 2: 擬似コード：検索アルゴリズム

- nR_{level} : ノード n のレベル $level$ の右隣接ノード．
- nL_{level} : ノード n のレベル $level$ の左隣接ノード．
- $n.key$: ノード n のキー．

ノード s は自身の所属する最上位のリストから検索を開始する．まずノード s は s と d を比較し，$s = d$ ならば検索終了する (2〜3 行目)．$s \neq d$ の場合，$s > d$ ならばリストを左方向に，$s < d$ ならばリストを右方向に辿り，隣接ノードへとクエリを転送する (6〜8，12〜14 行目)．

隣接ノードでも同様に処理を行い，リスト中で d を超えないノードのうち，最も近いノードまで来た場合，一つ下のリストに降り，以下同じ処理を繰り返す (9，15 行目)．この検索方法で，レベル 0 のリストまで降りることでシステム内に存在するノードは確実に発見できる．システム内に該当ノードが無い場合は，検索キーにもっとも近いキーを持つノードを結果として返す (16〜17 行目)．

検索に要する時間の期待値は $O(\log_k N)$ となることが証明されている [2]．同様に新規ノードの挿入・既存ノードの削除等も $O(\log_k N)$ で実行可能である．

4 探索頻度偏りを考慮したスキップグラフの拡張

スキップグラフでは各資源は対等の関係にあり，どの資源を検索する場合も $O(\log_k N)$ の時間を要する．しかし実際のシステムにおいては，人気のある資源とそうでない資源の検索頻度には大きな格差がある場合

がほとんどである．その場合，人気のない資源の検索時間を多少犠牲にしても，人気のある資源の検索時間を減らすことでシステム全体としてパフォーマンスを向上させることが可能である．そこで，本研究では資源のアクセス頻度に偏りがあるようなスキップグラフ上において，ノードの扱いに格差をつけることでシステム全体のコストを最適化する手法を提案する．各ノードには自身の検索頻度に応じて重みというパラメータが与えられる．提案手法では重みの大きなノードほど高速に検索可能である．

4.1 重みに応じた検索時間の達成方法

提案手法の基本的なアイデアは，各ノードの重みに応じメンバシップベクタを複数割り当てることで，重みの大きな資源を多数のリストに出現させることである．以下提案手法における重み付けの方法の詳細について述べる．

スキップグラフでは，ノード x はメンバシップベクタ $m(x)$ をただ 1 つ持ち，その接頭部に従って所属リストが決定される．提案手法では，ノード x には重み $w(x)$ (非負整数) が与えられており，ノード x は自身を含めて $w(x)$ 個の複製を持つ．以下，ノード x の全複製集合を $grp(x)$ と記述し，**ノードグループ** $grp(x)$ と呼ぶ．さらに各複製を区別するため，ノード x_i と書いてノードグループ x の i 個目の複製を表すこととする．複製されたノードはそれぞれ独立にメンバシップベクタを持ち，個々のメンバシップベクタに応じて各レベルの所属リストを決定し，各々の隣接ノードを管理する．それにより，スキップグラフではどのノードも各レベルにおいてただ 1 つのリストに出現するのに対し，提案手法では，ノードグループ x は同レベルのリストのうち最大 $w(x)$ 個のリストに出現する．提案手法に基づいて拡張を施したスキップグラフを以降**重み付きスキップグラフ**と呼ぶ．図 3 に重み付きスキップグラフの例を示す．

同ノードグループに属するノードはいずれも同じ情報資源を管理しているため，探索の際はノードグループ内のいずれかのノードを発見すればよい．

4.2 探索時間の見積もり

本節では重み付きスキップグラフにおいて，W を全ノードの重み総和としたとき，ノードグループ g を探索する際の期待実行時間が $O(\log_k(W/w(g)))$ となることを示す．証明の際には新たに以下のような表記を使用する．

- $w(g)$：ノードグループ g の複製数．
- N_G：システム内のノードグループ総数．
- $\mathbb{G}$：全ノードグループ集合．$(|\mathbb{G}| = N_G)$

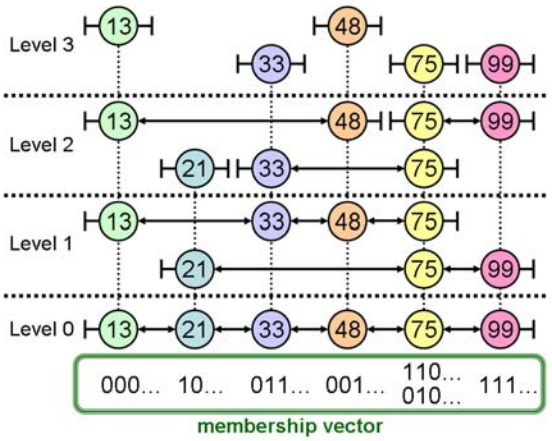

図 3: ノードグループ 75 を 2 個に多重化した重み付きスキップグラフ

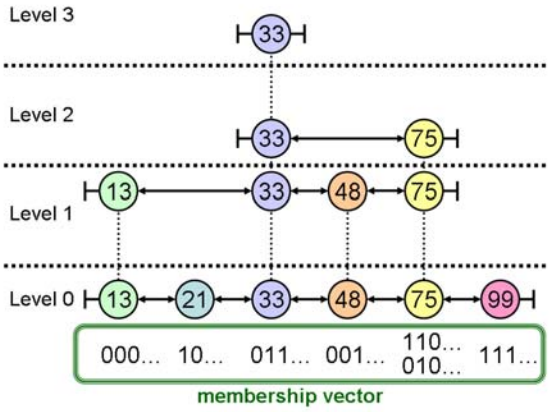

図 4: 図 3 を基にした，各レベルについてノード 33 を含むリストのみ出現する部分グラフの例

以下，検索クエリを生起したノードを s_i，検索目標となったノードグループを D とする．

定理 1. 重み付きスキップグラフにおいて，ノードグループ D の期待検索時間は $O(\log_k \frac{W}{w(D)})$ となる．

証明（概略）．検索アルゴリズムより，検索の際はノードグループ D のノードのうちいずれかを発見すればよい．また，探索経路に含まれる可能性のあるリストはノード s_i の所属するリストのみである．よって図 4 に示すように，各レベルについて s_i を含むリスト 1 つのみが出現するような部分グラフ上で考える．

初めにノード s_i の最上位レベルについて考える．$maxLevel_{s_i}$ をノード s_i の所属するリストのうち最上位リストのレベルとすると，$maxLevel_{s_i}$ はノード s_i が初めて孤立するレベルとなる．メンバシップベクタの各文字は独立かつ一様ランダムに選択されるため，

レベル i のリストに出現するノード数を N_i とすると，

$$E(N_i) = 1/k \cdot N_{i-1}$$

が成り立つ．$E(maxLevel_{s_i})$ は $E(N_i) = 1$ となるレベルと考えられるため，

$$E(maxLevel_{s_i}) = \log_k W$$

となる．

次に D に含まれるノードがどのレベルのリストまで出現するかについて考える．$maxHeight_D$ を D に含まれるノードの中で最も高いレベルまで出現するノードの出現レベルとする．$maxLevel_{s_i}$ と同様に考えると，

$$E(maxHeight_D) \quad = \quad \log_k w(D)$$

となる．

よって部分グラフ上における s_i と D の最大レベルの差の期待値は

$$E(maxLevel_{s_i}) - E(maxHeight_D) = \log_k \frac{W}{w(D)}$$

となる．また，1 レベルあたり辿るべきリンク数は確率 $1/k$ の幾何分布と考えられるため，その期待値は

$$\frac{1-1/k}{1/k} = k - 1$$

となり，これは定数である．

ゆえにノードグループ D の検索に要する時間の期待値は $O(\log_k \frac{W}{w(D)})$ である． □

4.3 最適な重み付け手法

本節では，ノードグループ i の単位時間当たりの検索回数 $s(i)$ が既知であるとき，各資源に対してどのように重み付けをするとシステム全体として検索に要する通信コスト (**総検索時間**と呼ぶ) が最小となるかについて考える．

最初に総検索時間について定義する．

定義 1. 総検索時間 L は以下の式で定義される．

$$L = \sum_{i \in \mathbb{G}} (s(i) \text{ 回のノード } i \text{ 検索に要した時間の総和})$$

定理 1 より，ノードグループ i の検索時間は $O(\log_k W/w(i))$ となるため，L の期待値 $E(L)$ は漸化的には以下のように定義できる．

$$E(L) = \sum_{i \in \mathbb{G}} s(i) \log_k W/w(i)$$

$E(L)$ に関して次の定理が成り立つ．

定理 2. $\{a, b, c, \cdots\} \in \mathbb{G}$ とした時，$E(L)$ を最小化するような重み付け方法は

$$w(a) : w(b) : w(c) : \cdots = s(a) : s(b) : s(c) : \cdots$$

である．

証明. 一般性を失うことなく，各ノードグループを検索キーに関して昇順にソートした列の先頭から，ノードグループ $1, 2, \cdots, N_G$ と名前付けができる．すると目的関数 $E(L)$ は

$$E(L) = \sum_{i=1}^{N_G} s(i) \log_k W/w(i)$$

と表せる．

多変数関数において，最小値を取る点の候補には極小点と境界点の 2 種類が存在する．

まず境界点について考える．理論上 $w(i)$ は非負整数であるという以外に制限は無いため，$w(i)$ の取りうる値の範囲は $[0, \infty)$ となる．このとき境界点は $w(i) = 0$ と $w(i) = \infty$ の 2 つである．

1. $w(i) = 0$ のとき
   $w(j)$ $(j \neq i)$ を定数として考えると $W = w(i) + \epsilon$ (ϵ は定数) と表現できるので，

   $$\lim_{w(i) \to +0} log_k \frac{W}{w(i)} = \lim_{w(i) \to +0} log_k \frac{w(i)+\epsilon}{w(i)} = \infty$$

   となる．ゆえに $E(L)$ は，

   $$\begin{aligned} \lim_{w(i) \to +0} E(L) \quad &= \quad s(1) \log_k \tfrac{W}{w(1)} + \cdots + s(i) \cdot \infty \\ &\quad + \cdots + s(N_G) \cdot \log_k \tfrac{W}{w(N_G)} \\ &= \quad \infty \end{aligned}$$

   $s(i) = 0$ $(0 \leq i \leq N_G)$ であれば $E(L)$ は定数値であるが，その場合 $w(i) = \alpha s(i)$．よってその場合も定理 2 を満たす．

2. $w(i) = \infty$ のとき
   $\lim_{w(i) \to \infty} W = \infty$ より，

   $$\lim_{w(i) \to \infty} E(L) = s(1) \cdot \infty + \cdots + s(N_G) \cdot \infty = \infty$$

よって $s(i) = 0$ である場合を除き，境界点では最小値を取らない．

次に極小点について考える．極小点の候補となるのは臨界点であるため，$E(L)$ の臨界点 $(w(1), w(2), w(3), \cdots, w(N_G))$ を求める．$E(L)$ を $w(i)$ について偏微分すると

$$\frac{\partial}{\partial w(i)} E(L) = \frac{s(1) + s(2) + \cdots + s(N_G)}{w(1) + w(2) + \cdots + w(N_G)} - \frac{s(i)}{w(i)}$$

臨界点では全ての変数について 1 階偏微分が 0 となる [12] ため，全ての $w(i)$ $(i = 1, 2, \cdots, N_G)$ について 1 階偏微分を計算し，連立方程式を作ると，

$$\begin{cases} \frac{s(1)+s(2)+\cdots+s(N_G)}{w(1)+w(2)+\cdots+w(N_G)} = \frac{s(1)}{w(1)} \\ \vdots \\ \frac{s(1)+s(2)+\cdots+s(N_G)}{w(1)+w(2)+\cdots+w(N_G)} = \frac{s(N_G)}{w(N_G)} \end{cases}$$

左辺は全ての $w(i)$ について同じであるため，$\frac{s(1)+s(2)+\cdots+s(N_G)}{w(1)+w(2)+\cdots+w(N_G)} = \frac{1}{\alpha}$ とおいて整理すると

$$\begin{cases} w(1) = \alpha s(1) \\ \vdots \\ w(N_G) = \alpha s(N_G) \end{cases}$$

よって臨界点は存在し，$(\alpha s(1), \alpha s(2), \cdots, \alpha s(N_G))$.

定義 1 より，α がどのような値でも $E(L)$ の値は変わらない．よって α を変化させる事により作る事のできる複数の臨界点において，$E(L)$ の値は等しい．

2 つの境界点では双方とも $E(L)$ の値は無限大に発散する事，臨界点は等しい実数値を取ることより，すべての臨界点において最小値を取ることが分かる．

よって，定理 2 は成り立つ． □

5 評価

本節では，前節で提案した重み付きスキップグラフについて，シミュレーションにより評価を行う．以降では検索頻度の高い順にノードグループ $1, 2, \cdots, N_G$ と呼ぶ．

5.1 重みの制限方法

4.3 節では，最適な重み付け手法を考案した．しかし分散データ構造として実装する場合，ノードの重みの最小単位は 1 となるため，検索頻度に極端な差がある際に正しい比を保った場合，一部の人気ノードの重みが莫大なものとなる．これはメンテナンスコストの観点から実用的とは言えない．よって本節では，現実的なシステムへと適用可能な重み付けの方法を示す．

5.1.1 重みの上限 · 下限の設定

最適な重み付けでは，ノードの検索回数に比例した重みを割り当てていた．しかし，同一のノードグループを無限に複製可能とするのはリンク数の爆発を招くため，メンテナンスコストの問題から現実的ではない．よって複製の個数に上限 MAX を設定する．また，一度も検索対象となっていないノードグループについても複製は必ず 1 つ以上持つとする．

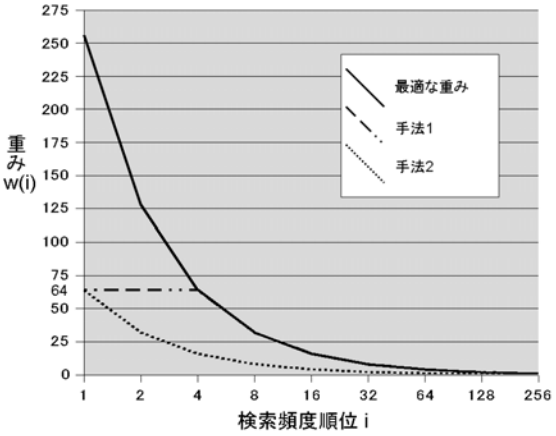

図 5: 重み付けの例：$MAX = 64$ の場合

以上よりスケーリングを施した場合，ノードグループ x について重み $w(x)$ の定義域は $[1, MAX]$ と再定義される．

5.1.2 スケーリング法

本節では，前節で制限した重みの定義域内に，理想値をスケーリングする方法について述べる．ここでは *CutOff*，*Scaling* と呼ばれる 2 種類の手法を提案する．

[手法 1:*CutOff*]
基本的に重み=検索頻度として割り当てるが，上限を超える場合または下限を下回る場合は余剰分をカットし，定義域内に収める方法．ノードグループ x への重みの割り当ては以下のように定義される．

$$w(x) = \begin{cases} 1 & (s(x) < 1) \\ s(x) & (1 \leq s(x) \leq MAX) \\ MAX & (s(x) > MAX) \end{cases}$$

[手法 2:*Scaling*]
検索回数の比が $[1, MAX]$ の範囲に収まらない場合に重みの定義域内に検索回数の比を圧縮し，圧縮した値を重みとして割り当てる方法．ノードグループ x への重みの割り当ては以下のように定義される．

$$w(x) = \begin{cases} 1 & (s(x) < 1) \\ s(x) & (1 \leq s(1) \leq MAX) \\ \lceil \frac{s(x)}{s(1)} \times MAX \rceil & (s(1) > MAX) \end{cases}$$

定義から分かるとおり，手法 2 には $s(1)$(検索回数の最も多いノードグループの検索回数) が必要である．分散環境において $s(1)$ を取得する方法については紙面の都合上割愛する．最適な重み付けに対し，手法 1 で重み付け，手法 2 で重み付けを施した場合の例を図 5 に示す．

5.2 シミュレーション結果

5.2.1 総検索時間の比較

従来のスキップグラフ，最適な重み付きスキップグラフ，手法 1 で重み付けしたスキップグラフ，手法 2 で重み付けしたスキップグラフの 4 つのスキップグラフについて，シミュレーションにより総検索時間とメンテナンスコストの比較を行う．本節における全てのシミュレーションは以下のようなパラメータ設定のもとで行われている．

- メンバシップベクタ $\{0,1\}$ の文字列
- ノードグループ数 $N_G = 1,024$
- 重み上限 $MAX = 256$
- 検索クエリ数 $100,000$

検索クエリを生起するノードはシステム内から一様ランダムに選択される．検索対象となるノードの選択確率は Zipf 関数[1] に従う．すなわち，検索頻度の高い順にノードグループ $1, 2, \cdots, N_G$ とおいた場合，ノードグループ x が検索対象として選択される確率 $Pr(D = x)$ は，以下のようになる．

$$Pr(D = x) = \frac{x^{-\alpha}}{\sum_{i=1}^{N_G} i^{-\alpha}}$$

よって最適な重み付けを施した場合各ノードの重みは

$$(w(1), w(2), \cdots, w(N_G)) = (\lceil \frac{1}{N_G^{-\alpha}} \rceil, \lceil \frac{2^{-\alpha}}{N_G^{-\alpha}} \rceil, \cdots, 1)$$

となる．重み付け手法による総検索時間の違いに注目するため，本節におけるシミュレーションでは静的なシステムを仮定する．すなわち，各ノードの検索確率はシミュレーション中に変化する事は無く，ノードの出入りも起きないとする．

以上のような設定のもとで 4 種類のスキップグラフに対し行ったシミュレーションにおける総検索時間の比較結果を図 6 に示す．Zipf 係数 α を $[0.5, 1.5]$ の範囲で 0.1 刻みに変化させ，各 Zipf 係数につき 100 回の試行を行った平均の値を示している．

シミュレーション結果より，最適な重み付けを施した場合，α の値が（検索頻度頻度偏りが）大きくなればなるほど L が減少している事が分かる．手法 1 では $\alpha = 0.8$ から最適な場合と差が出始め，$\alpha = 1.2$〜1.5 では L はほぼ同じという結果が出ている．これは，手法 1 では単純に余剰分をカットしているため，ノード 1 とそれ以外の上位ノード間には大きな差があるにもかかわらず同等の重みしか割り当てられないためと考えられる．それに対し手法 2 では $\alpha = 0.8$〜1.2 の範囲において手法 1 に比べ L の値が大きいが，$\alpha = 1.3$〜1.5 では逆に手法 1 より L の値は小さくなり，かつ $\alpha = 1.2$ 以降も減少が続いている．以上のことより，比を圧縮してでもノード間の重みに格差を付けることが有効であると分かる．

[1] k 番目に頻度の多い事象の起こる回数が $k^{-\alpha}$(α:Zipf 係数) に比例するような逆冪乗分布．頻度分布の多くがこの分布に従う [5, 10].

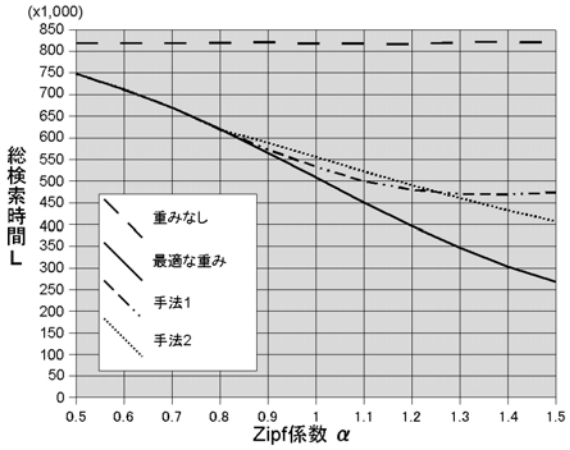

図 6: 総検索時間の比較

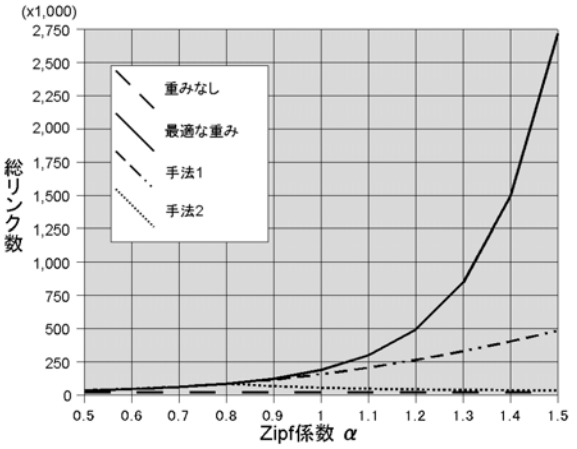

図 7: メンテナンスコストの比較

5.2.2 メンテナンスコストの比較

本節では重み付けを施した場合のメンテナンスコストについてシミュレーションによって検証する．メンテナンスコストは，システム内に存在する全ノードのリンク数総和によって評価する．前節と同じ設定でシミュレーションを実施した結果を図 7 に示す．

シミュレーション結果より，全ノードのリンク数総和について次のことが分かる．$\alpha = 0.5$〜0.8 は提案した 3 手法の間にほとんど差は見られない．これは最適な重み付けを施した場合の最大の重み $w(1)$ が $MAX(= 64)$ を超えないためである．しかし $\alpha = 0.9$ から手法 2 のリンク数総和が減り始める．これは比を定められた範

囲に圧縮しているため，偏りが大きくなるほどノードグループ 2 以降の重みが減っていくこととなるためである．

手法 1 では最適な重み付けを施した場合に比べ半分程度のリンク数総和にとどまっているが，手法 2 では α が大きければ大きいほどリンク数総和の減少効果も大きく，通常のスキップグラフと比べてもリンク数総和は 1.5 倍程度で抑えられているので，5.2.1 節の結果も合わせて考慮すると重み付けを行うことの有効性が分かる．

6 むすび

本研究では，スキップグラフの各ノードに重みを定義し，重みに応じた個数の複製を作る事で各ノードの検索時間を調整可能とする手法を提案した．提案手法では，ノード i の重みを $w(i)$，システム内の全ノード重み総和を W としたときに，ノード i の期待検索時間が $O(\log(W/w(i)))$ となる．さらに検索頻度が既知の場合に最適な総検索時間を実現する重み付け手法を提案し，その正当性を示した．この重み付け手法は理論的には最適であるが，メンテナンスコストの観点から実用上問題があるため，メンテナンスコストを削減するために手法 *CutOff* と手法 *Scaling* を提案し，シミュレーションにより総検索時間とメンテナンスコストの評価を行った．その結果 *CutOff* では検索頻度の偏りが少ない場合総検索時間において良好な結果を示すが，偏りが大きくなると性能が悪くなり，メンテナンスコストについても比較的大きくなる傾向が見られた．それに対して *Scaling* では，手法の実現のために必要な情報は *CutOff* より多いものの，全般的に総検索時間，メンテナンスコスト双方において良好な結果を示した．

謝辞— 本研究の一部は，文部科学省 21 世紀 COE プログラム（研究拠点形成費補助金）の研究助成，日本学術振興会科学研究費補助金（基盤研究 (B)15300017，若手 (B)15700017），文部科学省科学研究費補助金（特定領域研究 16092215），および，総務省戦略的情報通信研究開発推進制度（SCOPE）によるものである．

参考文献

[1] James Aspnes, Jonathan Kirsch, and Arvind Krishnamurthy. Load balancing and locality in range-queriable data structures. In *23rd ACM Symposium on Principles of Distributed Computing*, pages 115–124, July 2004.

[2] James Aspnes and Gauri Shah. Skip graphs. In *14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 384–393, January 2003.

[3] Amitabha Bagchi, Adam L. Buchsbaum, and Michael T. Goodrich. Biased skip lists. In *13th Annual International Symposium on Algorithms and Computation*, pages 1–13, November 2002.

[4] Nicholas Harvey, Michael B. Jones, Stefan Saroiu, Marvin Theimer, and Alec Wolman. Skipnet: A scalable overlay network with practical locality properties. In *Proc. of the 4th USENIX Symposium on Internet Technologies and Systems*, March 2003.

[5] Adriana Iamnitchi, Matei Ripeanu, and Ian T. Foster. Locating data in (small-world?) peer-to-peer scientific collaborations. In *1st International Workshop on Peer-to-Peer Systems*, pages 232–241, March 2002.

[6] Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Proc. of the ACM SIGCOMM 2001*, pages 149–160, August 2001.

[7] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, August 1989.

[8] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM 2001*, pages 161–172. ACM Press, August 2001.

[9] Antony Rowstron and Peter Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. of the IFIP/ACM International Conference on Distributed System Platforms (Middleware)*, pages 329–350, November 2001.

[10] Kunwadee Sripanidkulchai. The popularity of gnutella queries and its implications on scalability. URL `http://www.cs.cmu.edu/~kunwadee/research/p2p/paper.html`, 2001.

[11] Ben Y. Zhao, John Kubiatowicz, and Anthony D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, Department of Computer Science, University of California, Berkeley, April 2001.

[12] 黒田 成俊. **共立講座 *21* 世紀の数学第 *1* 巻 微分積分**, chapter 8. 共立出版, 2002.