P2P ネットワークにおけるスキップグラフと接尾辞配列を用いた部分一致検索手法

坂野 遼平　佐藤 晴彦　小山 聡　栗原 正仁

構造化オーバレイは，大規模なノード群におけるノード同士の自律分散的な相互検索を実現する技術であり，特に分散ハッシュテーブルとしての活用で知られている．しかし，従来の構造化オーバレイでは完全一致検索以外の高度な検索機能を実現し難いことが弱点とされてきた．部分一致検索もそのひとつであり，今までにキーを n-gram に分割する等いくつかの手法が提案されてきたが，それら既存手法では部分文字列の出現頻度偏りに弱くノード毎の負荷が偏ってしまう等の問題があった．そこで本稿では，範囲検索が可能な構造化オーバレイである Skip Graph に，文字列検索に適したデータ構造である接尾辞配列の概念を組み合わせた新しい部分一致検索手法を提案する．提案手法及び既存手法の双方についてシミュレーション実験を実施することで，提案手法がノード数 N に対し $O(\log N)$ の検索時間を達成しつつ，従来手法で問題となっていたノード負荷の偏りを解消する等多くの利点を持つことを確認した．

Structured overlay networks, known for the application to distributed hash tables, provide searching for nodes without any centralized indexes in large scale networks. However it is hard to handle complex queries represented by partial match queries. There are past studies trying to realize partial match retrieval on structured overlay networks such as dividing keys into n-grams. However, those studies have serious problems e.g. affected by the bias of the n-gram frequency. In this paper, we propose a new partial match retrieval method which combines Skip Graph, a structured overlay network supporting range queries, and Suffix Array, a useful data structure in string manipulation. We report on the results of simulation experiments and describe that the proposed method can handle partial match queries within $O(\log N)$ overlay routing hops for a network of N nodes. We also show that the proposed method is free from various problems including load imbalance among nodes.

1　はじめに

Peer-to-Peer(P2P) システムは，ノード同士の自律的な協調動作によって資源を分散管理し，様々な機能を提供する分散型のシステムである．サーバによる情報の集中管理を行うシステムと比べ，単一障害点を排除でき，サーバの運用コストが不要になる等の利点があり，注目を集めている．

資源を分散管理することから，P2P システムにおいては，利用したい資源の管理を行なっているノードを発見可能とする技術が必要となる．初期の P2P システムではフラッディング等の手法が用いられていたが，近年ではスケーラビリティの高い構造化オーバレイが注目を集めている．

構造化オーバレイは，P2P ネットワークにおけるノード毎の経路表の構築や管理の方法を規定することでノード間のルーティングを実現する技術である．アプリケーション層マルチキャスト [4] や，ユビキタスネットワークのためのプラットフォームへの応用 [13] 等，研究・開発が盛んであり，特に分散ハッシュテーブル (DHT) [7][9][10] としての活用で知られている．

A Partial Match Retrieval Method on P2P Networks using Skip Graph and Suffix Array.

Ryohei Banno, 北海道大学大学院情報科学研究科 (現在, 日本電信電話株式会社 NTT 未来ねっと研究所), Graduate School of Information Science and Technology, Hokkaido University (Presently with NTT Network Innovation Laboratories, Nippon Telegraph and Telephone Corporation).

Haruhiko Sato, Satoshi Oyama, Masahito Kurihara, 北海道大学大学院情報科学研究科, Graduate School of Information Science and Technology, Hokkaido University.

コンピュータソフトウェア, Vol.29, No.3 (2012), pp.164–180.
[研究論文] 2011 年 10 月 20 日受付.

DHT では，各ノードに ID が割り当てられ，構造化オーバレイがノードの ID に基づくルーティングを担うことで，主に以下 2 つの機能を提供することを実現している．

put(key, value) キーと値のペアをオーバレイ上に登録する．

get(key) キーを指定してオーバレイから値を取り出す．

すなわち，DHT ではキーを指定することで，そのキーを担当するノードを一意に特定し値の格納・取得を行うことが可能である．しかし，値を取得する際のキーは完全一致による指定しか行えず，キーの部分文字列を指定する等の柔軟な検索はできない．

このように，一般に構造化オーバレイにより実現されるノード検索においては，柔軟なクエリを扱い難いという問題がある．本稿では，P2P ネットワークに参加する各ノードが予め 1 つあるいは複数のラベルを持っている状況を想定し[†1]，ネットワーク内のノードがラベルの指定により他のノードを検索するという問題設定について議論する．

なお，上記ラベルについて本稿ではノードキーと呼ぶこととし，DHT において一般にキーと呼ばれるものと以下のように区別する．

キー DHT において，格納または取得する値の指定に用いられる情報．

ノードキー 各ノードが，他のノードから自身を検索するための検索対象として保持するラベル．

DHT の場合，ノードキーをキーとし，ノードキー保持ノードへのポインタを値として put しておくことで，ノードキーの指定によるノード検索を実現できるが，前述のとおりノードキーの指定については完全一致が必要となる．

本稿では，範囲検索が可能な構造化オーバレイである Skip Graph [1] を用いて，ノードキーの部分一致検索を実現する手法を提案する．なお本稿では，部分一致検索及び関連する概念を以下のように定義する．

- 各ノードはノードキーを 1 つ以上持つ．
- ノードキーとして，本稿ではファイル名等の一般的な文字列を想定するが，次に示す部分文字列の定義が適用可能であれば任意のシンボルの列で良い．
- ノードキー nk が文字の列 $a_1a_2\cdots a_m$ で表されるとき，nk の部分文字列の集合 S_{nk} は，$\{a_ia_{i+1}\cdots a_j \mid 1 \leq i \leq j \leq m\}$ である．
- 有限長の文字列の集合を S とした時，部分一致検索を以下のように定義する．

  “ある文字列 $s \in S$ で検索を行った時，$s \in S_{nk}$ であるようなノードキー nk を持つ全てのノードにクエリが配信される仕組み．”
- 上記部分一致検索の定義において，$s \in S_{nk}$ であるようなノードキー nk を持たないノードにクエリが配信されても良いが，その場合，そのクエリが自分宛ではないことを判断できるものとする．

構造化オーバレイにおける既存の部分一致検索手法では，ノードによって負荷が大きく異なってしまったり，検索結果に偽陽性が生じる等の問題があった．提案手法では，文字列検索に適したデータ構造である接尾辞配列 [6] の概念を Skip Graph と組み合わせることで，$O(\log N)$ の検索ホップ数を達成しつつ，従来手法における諸々の問題点を解消することを可能としている．

以降，2 章では構造化オーバレイにおける部分一致検索に関する既存研究とその問題点について述べる．3 章で提案手法の説明を行い，4 章でシミュレーション実験の結果を交えて提案手法の特徴と優位性について説明する．最後に，5 章で総括する．

2 関連研究

2.1 既存の部分一致検索手法

構造化オーバレイにおける部分一致検索については，従来から研究が試みられている．主に，

1. 複数の完全一致検索処理を組みあわせることで部分一致検索を実現するもの，
2. 構造化オーバレイ上に部分一致検索可能なデータ構造を構築するもの，
3. Bloom filter [3] を用いて圧縮したインデックス

†1 例えばファイル共有システムを想定した場合，各ノードの保持するファイルのファイル名がそのノードのラベルとなる．

を各ノードが保持するもの，
等が存在する．ここではそれぞれの代表的な既存手法について概説する．

2.1.1 PIER

PIER [5] は大規模な分散クエリエンジンであり，DHT 上に登録されたドキュメントの ID を，ドキュメント名に対する部分一致検索で取得する手法を提案している．PIER ではドキュメント名を n-gram に分割して DHT 上にポインタを登録することで部分一致検索を実現する．

例えば “JAPAN” という名前のドキュメントを DHT に登録する場合，ドキュメント名を 2-gram に分割すると “JA”, “AP”, “PA”, “AN” の 4 個が得られる．これら 2-gram をキーとし，ドキュメントの ID を値として DHT に登録しておく (図 1)．検索を行う時は，検索文字列を n-gram(上例では 2-gram) に分割し，各 n-gram で lookup を行う．これにより各 n-gram に関連付けられたドキュメント ID のリストが複数得られる．そのリストの集合からドキュメント ID の出現回数を調べ，検索文字列の n-gram の数と比較することで，マッチするドキュメント ID を特定する．

ドキュメント名をノードキーと置き換えれば，PIER の手法によりノードキーに対する部分一致検索が可能である．即ち，各ノードは自身のノードキーをキーとし，IP アドレス等を値として予め put しておく．PIER の手法によりノードキーの ID が得られれば，その ID から IP アドレス等を取得することで，ノードキー保持ノードにメッセージを届けることが可能となる．

2.1.2 DST

DST [12] は，P2P 上に保管されるリソースに説明文が付与されていることを想定し，説明文に対する部分一致検索によってリソースのキーを発見可能とする．

DST では，DHT 上に接尾辞木 [11] を構築する．接尾辞木は文字列の接尾辞を木構造で表したデータ構造である．各エッジには文字列が対応付けられており，根から葉までの経路がひとつの接尾辞を表す．例えば文字列 “ababa” の接尾辞木は，図 2 のようになる．

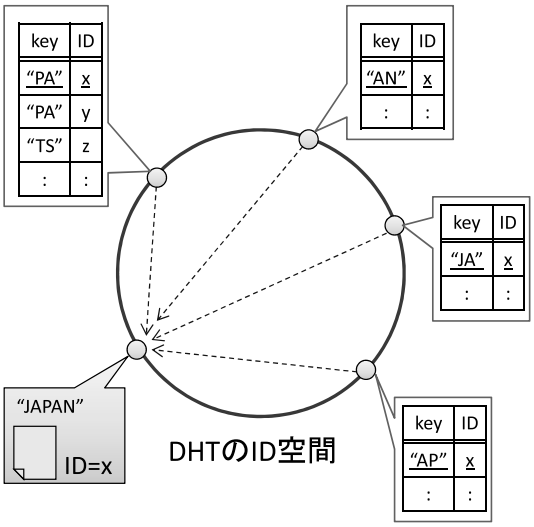

図 1 PIER における n-gram をキーとしたポインタ登録

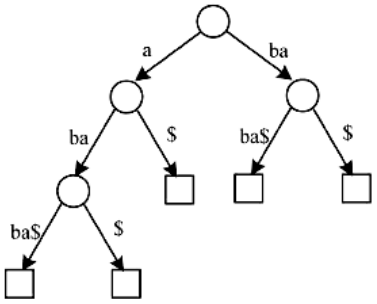

図 2 文字列 “ababa” の接尾辞木．論文 [12] の Fig. 2 の一部を引用

なお，“$” は終端記号であり，終端エッジ (終端ノードとその親ノードを結ぶエッジ) には，“$” か，末尾に “$” を伴う文字列が対応付けられることになる．

DST ではこの接尾辞木のルートエッジ (ルートノードとその子ノードを結ぶエッジ) を，接頭 1 字をキーとして DHT 上に登録する．この時，値として子エッジの ID 等を含むエントリを保持させる．ルートエッジ以外のエッジについては，親エッジから与えられた ID によって DHT 上に分散配置され，ルートエッジと同様に子エッジの ID 等を値として保持する．図 3 は図 2 の接尾辞木が与えられた場合の DST の構築例を示している．

検索を行う際は，まず検索文字列の接頭 1 字をキー

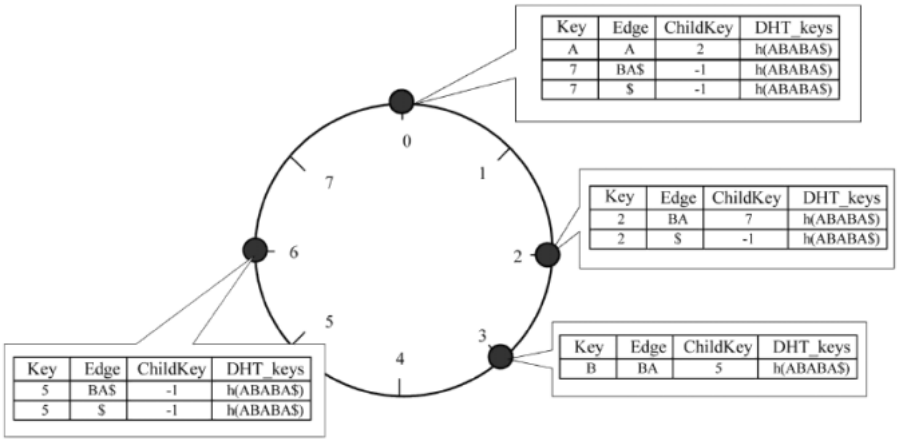

図 3 DST の構築例．論文 [12] の Fig. 5 より引用

として DHT 上で検索を行うことでルートエッジを特定し，それ以降は子エッジを辿っていく．

ノードの IP アドレス等をキーとし，ノードキーを前述の説明文として扱うことで，指定した部分文字列を含むノードキーを持つノードにメッセージを届けることが可能となる．

2.1.3 BF Skip Graph

Bloom filter [3] を用いた関連研究としては，BF Skip Graph [14] が挙げられる．BF Skip Graph は複数のキーワードによるノード検索を効率的に行うことができる手法である．Bloom filter を用いると，ある範囲のノード群に含まれるキーワード集合をコンパクトに表現できることを利用し，Skip Graph 上の各ノードがレベル毎に Bloom filter を保持することで複数キーワードによる絞り込み検索を実現している．Bloom filter にノードキーの n-gram や形態素を格納すれば，BF Skip Graph でもノードキーに対する部分一致検索が実現可能である．

2.2 既存手法の問題点

2.2.1 ノード負荷の偏り

PIER や DST のような DHT ベースの手法では，キーの出現頻度偏りに弱く，ノード毎の負荷に大きな差が出てしまうという問題がある．

例えば PIER の手法では，ノードキーの n-gram をキーとして DHT 上に登録するが，n-gram の出現頻度には一般に大きな偏りがある．図 4 は，Twitter から取得した 10,000 個のハッシュタグ[†2]を対象とし，2-gram の出現頻度を示したものである．なお，ここで用いた 10,000 個のハッシュタグの平均文字列長は約 8.7 文字であり，含まれる 2-gram は全部で 3,867 種類であった．このハッシュタグ集合は 4 章の実験においても用いるため，以降 *tags* と呼ぶことにする．このグラフからわかるように，ごく一部の 2-gram が 1,000 回前後出現する一方で，1 回しか出現しない 2-gram が 1,655 種類あり，出現頻度には大きな偏りがある．PIER では，同じ n-gram を含むノードキーへのポインタが全て DHT 上の同じノードに預けられるため，こういった出現頻度の偏りがそのままノード

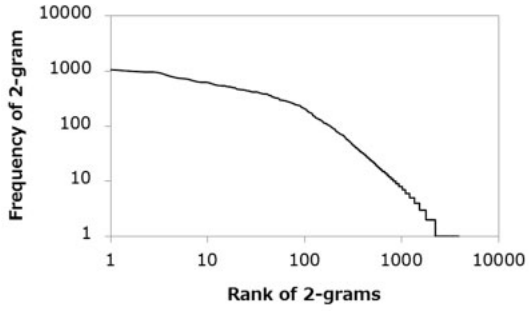

図 4 10,000 個のハッシュタグ群における 2-gram の出現頻度偏り

†2 Twitter 社が提供する StreamingAPI を用いて 2011 年 7 月 30 日に収集したもの．

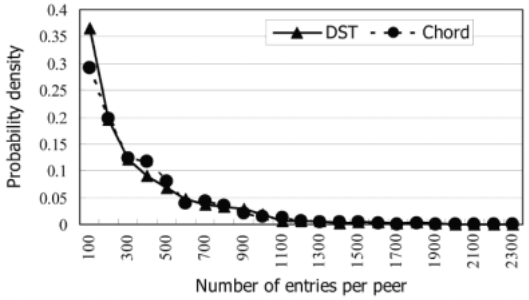

図 5　DST におけるノード毎のエントリ数に関する確率分布．論文 [12] の Fig. 9 より引用

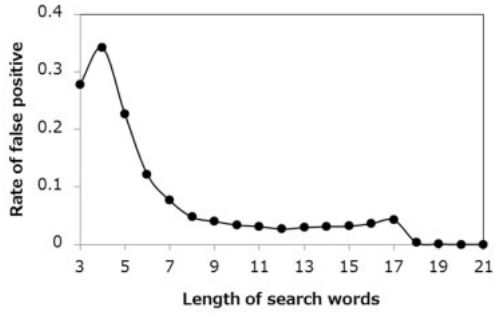

図 6　10,000 個のハッシュタグ群における偽陽性発生率

負荷の偏りに反映されてしまう．

また，DST においても，PIER と同様の問題が発生する．図 5 は DST 及びそのベースとなっている DHT(Chord) において，各ノードが保持しなければならないエントリの数に関する確率分布を示したものである．グラフから，一部のノードでエントリ数が非常に多くなっていることがわかる．

このような負荷の偏りは，通信量の偏りも発生させる．即ち，頻出する部分文字列は検索文字列においても頻出する可能性が高いため，出現頻度の高い部分文字列を担当するノードはメッセージの送受信を高頻度で行うことになり，他ノードと比べ通信負荷が高くなってしまう．

2.2.2　検索処理に要するメッセージ数

PIER の場合，検索文字列を n-gram に分割してそれぞれ検索処理を行うため，1 回の検索で検索文字列長に比例する通信が発生することになり，検索に伴う通信コストが大きいという問題がある．

2.2.3　検索文字列への制約

PIER で n-gram を用いた場合，n 文字未満の部分文字列は検索することができない．これは BF Skip Graph において Bloom filter にノードキーの n-gram を格納するようにした場合にも発生する問題である．

2.2.4　検索結果の偽陽性

n-gram を用いる手法では，検索で得られる結果に偽陽性が発生し得るという問題もある．これは，n-gram の順序や出現回数を考慮できないためである．例えば 2-gram を用いる場合，検索語 “sea” に対して “ease” 等のノードキーがマッチしてしまったり，検索語 “yoyo” に対して “toyo” 等のノードキーがマッチしてしまう．このことは無駄な通信の発生を伴うため，望ましくない．

図 6 は，*tags* を対象として PIER の手法を用いた場合の偽陽性発生率を示している．長さ l の検索語として，*tags* 内の各ハッシュタグの l-gram を用いた†3．PIER で 2-gram を用いた場合を想定し，検索にマッチするハッシュタグのうち，検索語を含んでいないものの割合を算出した．グラフから，特に検索文字列長が比較的短い場合には，検索結果に数十パーセントの割合で偽陽性が含まれていることがわかる．なお，偽陽性発生率は対象とするデータセットにより変化するため，必ずしも一般的な性質ではないことに留意されたい．

2.2.5　Bloom filter のサイズ

BF Skip Graph では，各ノードがレベル 0 では自ノードのみに関する Bloom filter を保持し，レベル i では，レベル i における右隣接ノードとの間にレベル $i-1$ で含まれているノードの，レベル 0 からレベル $i-1$ までの Bloom filter のビット単位 OR をとったものを保持する．従って，レベルが 0 から L まである場合，各ノードはレベル $i>0$ では $\frac{N}{2^{L-(i-1)}}$ 個のノードの情報を集約した Bloom filter を持つことになる．つまり，あるノードの最上位レベルにおける Bloom filter は Skip Graph 全体の半分のノードの情

†3　検索語の重複は排除した．即ち，同じ検索語による検索は 1 度のみである．

報を包含する．

Bloom filter は空間効率の良いデータ構造であるが，偽陽性が発生する可能性があり，その確率は格納する要素の数が多いほど大きくなる．Bloom filter に用いるビット列のサイズを b，ハッシュ関数の個数を k，格納する要素の個数を x とした時，偽陽性発生率 p は次式で表される．

$$p = (1 - e^{-\frac{kx}{b}})^k$$

b と x を定数とみなした場合，p を最小にする k は

$$k = \frac{b}{x} \ln 2$$

で与えられ，この時

$$p = \frac{1}{2^k}$$

となることが知られている．この式から b と x の関係は

$$b = -\frac{\ln p}{(\ln 2)^2} x$$

と表すことができる．従って，偽陽性発生率 p を一定以下に抑えたい場合，ビット列のサイズ b は x に比例して大きく設定する必要があることになる．

以下の議論では，BF Skip Graph において Bloom filter にノードキーの 2-gram を格納することを想定する．この場合，x の大きさは 2-gram の語彙数となる．日本語文字列について考えると，JIS X 0208 において規定されている文字数は 6,879 であるから，2-gram の総語彙数は 47,320,641 である．しかし，一般的に実データにおいてこれら全てが出現することは稀であり，扱われるデータの種類によって出現する文字列には偏りがある．

例えば，Wikipedia 日本語版[†4]の記事タイトル[†5]を 2-gram に切り分けた場合，語彙数は 451,378 であった．以降，この記事タイトル群を *titles* と呼ぶ．一方，前述の *tags* の場合，日本語がほぼ含まれておらず，大部分が英数字のみで構成されることから，語彙数は 3,867 と *titles* と比べ少なめである．

これらについて，最上位レベルでの偽陽性を 0.01 に抑える場合，フィルタサイズ b は，*titles* の場合およそ 257.9Kbyte，*tags* の場合およそ 2.3Kbyte となる．BF Skip Graph では，各ノードが保持する Bloom filter の総数は $O(\log N)$ であるから，各ノードで Bloom filter の保管に必要とされるメモリ量は $O(b \log N)$ である．

さらに，BF Skip Graph では各ノードの持つ Bloom filter を定期的に更新する必要があり，あるノードの挿入や離脱，ノードキーの変更等が全ノードに伝搬されるのは，各ノードが $O(\log N)$ 回だけ更新処理を実行した後になる．その各更新処理毎に，Bloom filter の集合 bf_{update} を保持したメッセージが $O(\log N)$ 回転送される．bf_{update} には転送毎に Bloom filter が追加され，$O(\log N)$ 回の転送のうち最後の転送時には，当該ノードが持つ全 Bloom filter の更新版が収集されていることになる．従って，ノードの挿入・離脱等が反映されるまでにネットワーク上を流れるメッセージのサイズの総量は $O(bN \log^3 N)$ となるため，特に上述した *titles* のようにフィルタサイズが比較的大きくなる場合には，この更新のコストが大きくなってしまうという問題がある．

以上からわかるように，フィルタサイズ b はトレードオフパラメータとなっている．つまり，b を十分に大きくした場合には各ノードのメモリ消費量が増え，定期的更新に伴う通信量も増大する．一方，b を小さく抑えた場合，偽陽性発生率 p が大きくなり，検索時の無駄な通信が増えてしまう．前述のように，扱うデータの種類によって適切なフィルタサイズ b は異なるが，文献 [3] においてはその決定指針については言及されていない．また，フィルタサイズは固定的に設定されるため，ネットワーク上で扱われる語彙の規模が大きく変化した場合に追従できないという問題も考えられる．

3 提案手法

ここでは，要素技術及び提案手法の説明を行う．

なお，本章で述べる Skip Graph 及び Multi-key Skip Graph において，ノード及び仮想ノードは一般にキーと呼ばれる情報を持つが，これは 1 章で述べた DHT におけるキーとは意味合いが異なっており，

†4 http://ja.wikipedia.org

†5 2012 年 2 月 18 日取得，総数 1,269,108.

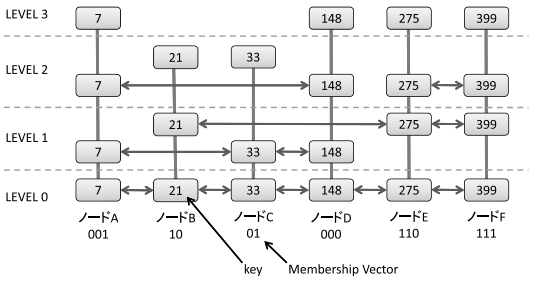

図 7　Skip Graph の構成例

本稿では以下のように定義する．

キー　Skip Graph 及び Multi-key Skip Graph において，物理ノード及び仮想ノードの挿入位置やリンク関係を定めるために用いられる情報．

またノードキーについては，1 章で述べた定義内の「ノード」という言葉は全て物理ノードを指しているものとする．

3.1　Skip Graph

Skip Graph [1] は，構造化オーバーレイ技術の一種である．全順序関係を持つキー列の上に多重にスキップリスト [8] を構築した構造を持っており，DHT と異なりキーをハッシュしないため，キーの範囲を指定した検索を行うことが可能である．

Skip Graph では，キーとそれを保持するノードは 1 対 1 に対応する．各ノードはキーの他に，ランダムに割り当てられる membership vector を持つ．membership vector は w 進数の整数であり，本稿では $w = 2$ とする．図 7 に，Skip Graph の構成例を示す．

Skip Graph は階層構造を成しており，各ノードは階層 (レベル) 毎にひとつのリストに参加する．レベル 0 では各ノードはキーの順にソートされ，隣接ノード間に双方向リンクが張られている．レベル i では，membership vector の接頭 i 桁までが等しいノード群によってリストが形成される．また，他ノードとのリンクが無く孤立した状態になるレベルがそのノードの最上位レベルとなる．以上より，レベル i には平均的に 2^i 個のリストが存在することになり，ノード数が N の時，Skip Graph のレベル数は $O(\log N)$ となる．従って，各ノードが保持しなければならない経路表のサイズも $O(\log N)$ となる．

Skip Graph における検索は，クエリを投げるノードの最上位レベルからスタートする．そして，スキップリストと同様に，目的キー以下の最大キーまでホップしてレベルをひとつ下げるという操作を繰り返す†6．上位レベルほど大きくスキップすることができるため，目的キーまで少ないホップ数で到達可能である．検索のホップ数は $O(\log N)$ となる．

3.2　Multi-key Skip Graph

通常の Skip Graph では，ノードとキーが 1 対 1 に対応しており，ノードはただひとつのキーしか保持することができない．Multi-key Skip Graph [15] は，ノードが複数のキーを保持できるように Skip Graph を拡張したものである．

Multi-key Skip Graph では，各ノード (物理ノード) がキー毎に仮想的なノード (仮想ノード) を用意し，この仮想ノードを Skip Graph に挿入することで，物理ノードの複数キー保持を可能とする．この時，仮想ノードの membership vector は物理ノードと同じものを用いる．即ち，membership vector は物理ノードに固有であり，各物理ノードはただひとつの membership vector を持つ．

通常の Skip Graph と同様のルーティングを行った場合，仮想ノード数に応じてホップ数が増大してしまう問題があるが，Multi-key Skip Graph ではマルチレンジフォワード方式によりこの問題を解消している．マルチレンジフォワード方式とは，各物理ノードが同じクエリを複数回受信することを防ぐために，保有する仮想ノードの経路表を用いて効率的にルーティングを行う方式である．これにより，単一キー検索，範囲検索のいずれにおいても，検索に要するホップ数の平均が物理ノード数 N に対し $O(\log N)$ となることを実現している．

3.3　接尾辞配列

接尾辞配列 [6] は，テキストの接尾辞を辞書順に

†6　目的キーが検索開始ノードのキーよりも大きい場合．

接尾辞	位置
a	10
abra	7
abracadabra	0
acadabra	3
adabra	5
bra	8
bracadabra	1
cadabra	4
dabra	6
ra	9
racadabra	2

図 8 文字列 “abracadabra” の接尾辞配列

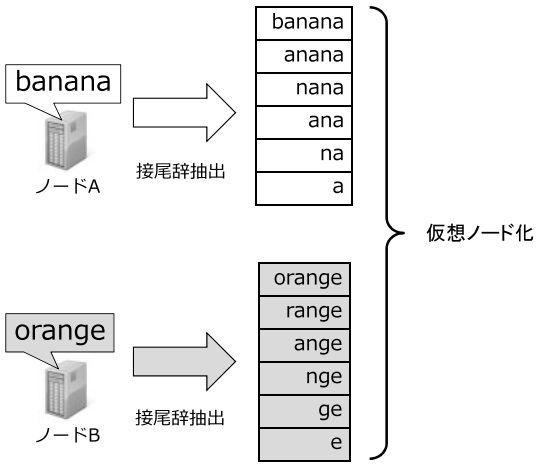

図 9 接尾辞の仮想ノード化

ソートした配列であり，主に文字列検索に用いられる．例えば，文字列 “abracadabra” の接尾辞配列は “bracadabra” や “racadabra” 等 11 個の接尾辞を元のテキストにおける出現位置と共に記録したものであり，図 8 のようになる．

あるテキストについて接尾辞配列が与えられた時，文字列の検索は 2 分探索によって高速に処理可能である．例えば上記 “abracadabra” について文字列 “ra” を検索する場合，まず接尾辞配列の中心要素 “bra” と比較する．辞書順で考えると “ra” が大きいため，接尾辞配列の後ろ半分に検索範囲が絞られる．このように検索範囲を絞る操作を繰り返すことで，最終的に “ra” にたどり着き，元テキストにおける出現位置を得ることができる．

3.4 Skip Graph と接尾辞配列を用いた部分一致検索

3.4.1 構築

本稿では，Skip Graph に接尾辞配列の概念を組み合わせた部分一致検索手法として Skip Suffix Array (SSA) を提案する．

SSA の基本アイデアは，各ノードのノードキーを接尾辞に分解し，それらひとつひとつを Multi-key Skip Graph の手法により仮想ノードとして挿入するというものである．例えば，ノードキーとして “banana” と “orange” の 2 つが存在する場合，図 9 のように

キー（接尾辞）	ノード
anana	A
ange	B
banana	A
e	B
ge	B
nana	A
nge	B
orange	B
range	B

辞書順

図 10 レベル 0 リストにおけるキー列と物理ノードの対応付け

なる．

仮想ノードはキー列の順序関係が保たれるように挿入されるため，レベル 0 リストのキー列はソートされた接尾辞の列となる．なお SSA では，キー列は数値も含め辞書順に並ぶことを想定している．レベル 0 リストのキー列とノードとの対応付けをまとめると，図 10 のように，接尾辞配列状になっていることがわかる．即ち，SSA は Skip Graph のレベル 0 リストにおけるキー列をソートされた接尾辞列とすることで，ネットワーク上に分散した接尾辞配列を形成するものであると言える．

なお，図 10 中でいくつかの接尾辞が省かれているのは，SSA では同一物理ノードが提供する接尾辞のうち，他の接尾辞のプレフィックスとなっているものを省略可能なためである．例えば “banana” の場合，

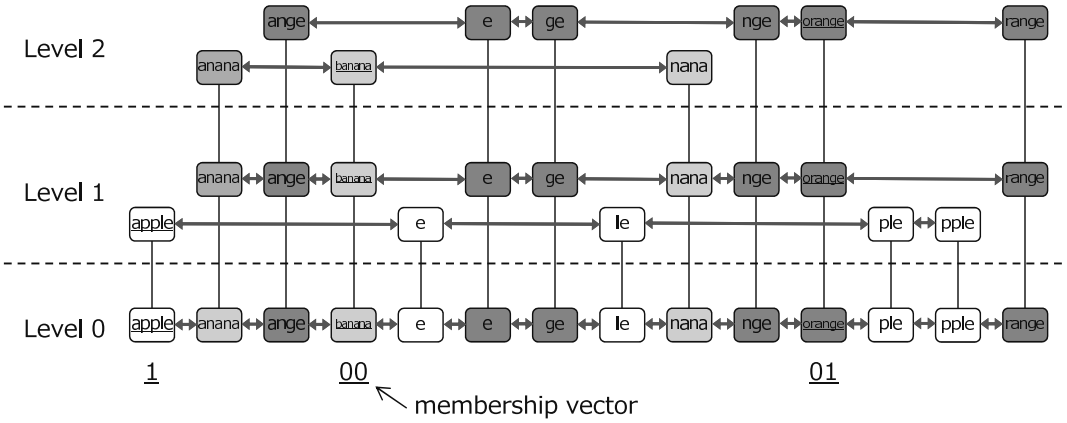

図 11　SSA のリンク構成

```
pn, physicalNode    物理ノード
localKeys    物理ノードが持つ元キーの集合
pn_intr    初期ノード
suffixes    物理ノードが持つ接尾辞キーの集合
vn, virtualNode    仮想ノード

getSuffixes()    キーの接尾辞の集合を求める処理
removeRedundantSuffix()
    冗長な接尾辞（他の接尾辞のプレフィックスである接尾辞）を除去する処理
createVirtualNode(key)    仮想ノードを作成する処理
join(pn_intr)    Multi-key Skip Graphにおけるノード加入処理

 1   pn.join(localKeys, pn_intr) {
 2       for each key in localKeys {
 3           suffixes.add(key.getSuffixes());
 4       }
 5       suffixes.removeRedundantSuffix();
 6       for each suffix in suffixes {
 7           vn = createVirtualNode(suffix);
 8           vn.join(pn_intr);
 9       }
10   }
```

図 12　ノード参加処理

```
pr, prefixRange    プレフィックスが一致する範囲

getKey()    仮想ノードが持つキーを取り出す処理
getPrefix()    prによって指定されているプレフィックス文字列を取り出す処理
isGreaterString(key)    辞書順で引数キーよりも大きいか否かを判定する処理
isLessString(key)    辞書順で引数キーよりも小さいか否かを判定する処理

 1   pr.contain(vn) {
 2       return vn.getKey().startWith(pr.getPrefix());
 3   }
 4
 5   pr.isGreater(vn) {
 6       return (!pr.contain(vn)
 7               && pr.getPrefix().isGreaterString(vn.getKey()));
 8   }
 9
10   pr.isLess(vn) {
11       return (!pr.contain(vn)
12               && pr.getPrefix().isLessString(vn.getKey()));
13   }
```

図 13　プレフィックス一致範囲 pr と仮想ノード vn の包含関係及び大小関係判定

接尾辞 “a”，“ana” は “anana” のプレフィックスであり，“a” や “ana” にマッチするクエリは必ず “anana” にもマッチするため，省略することができる．同様に，接尾辞 “na” も “nana” のプレフィックスになっているため省くことができる．

図 11 は，“apple”，“banana”，“orange” の 3 つのノードキーが存在する場合の，SSA のリンク構成を示している．図中で同じ明度のキーは同一物理ノードであり，各物理ノードはノードキーをひとつ保持している．

図 12 は物理ノード pn のオーバレイへの参加処理を示した擬似コードである．2 行目から 4 行目にかけて，保持している全てのノードキーを接尾辞へと分解し，*suffixes* へと格納している．5 行目で冗長な接尾辞の削減を行い，6 行目から 9 行目の処理で全接尾辞を仮想ノードとして Multi-key Skip Graph に挿入している．

3.4.2　部分一致検索処理

SSA における部分一致検索は，Multi-key Skip Graph におけるプレフィックス一致検索として実行される．すなわち，検索文字列をプレフィックスとして含む範囲を発見し，範囲内ノードにクエリをブロードキャストすれば良い．そのためには，プレフィックスが一致するキーの範囲を，通常の範囲検索[†7]処理で扱えるように定義する必要がある．Skip Graph の範囲検索処理における範囲 $range$ の要件は，指定したノードが $range$ に含まれるか否かを判定可能であることと，指定したノードと $range$ との大小関係を判

†7　以上，以下，超，未満の指定による範囲検索．

```
string   検索文字列
query    マッチするキーを持つノードに届けるクエリ

createPrefixRange(string)
    pr（引数文字列をプレフィックスとする範囲）を生成する処理
forwardQuery(pr, query)    Multi-key Skip Graphにおけるクエリ転送処理

1   pn.partialMatchSearch(string, query) {
2       pr = createPrefixRange(string);
3       forwardQuery(pr, query);
4   }
```

図 14 物理ノード pn における部分一致検索処理

定できることである．これらの判定は，図 13 の擬似コードに示した方法で実現できる．

従って，物理ノード pn における部分一致検索処理は，図 14 の擬似コードに示したように，Multi-key Skip Graph における検索処理に帰着させることができる．

例えば図 11 において，membership vector が 00 の物理ノードから，文字列 “ng” で検索した場合，キー “nana” を持つ仮想ノードの最上位レベルから検索を開始し，レベル 1 でリストを右側へたどってキー “nge” を発見し，membership vector が 01 のノードへとクエリが届くことになる．このように SSA は，接尾辞配列におけるスキップリストを用いた検索処理を，分散環境で実現したものとなっている．

4 提案手法の特徴と利点

本章では，シミュレーション実験の結果を交えて提案手法 SSA の特徴と利点について述べる．特に，2.2 節で述べた既存手法における種々の問題点が，SSA において解決されていることを示す．なお，以降において N は物理ノード数，M はノードキーの総数 (全ノードの保有するノードキー数の合計)，l_k はノードキーの平均文字列長，l_q は検索文字列の平均文字列長であるとする．

4.1 検索ホップ数

まず，検索時のホップ数について考える．SSA では，マルチレンジフォワード方式によってホップ数が $O(\log N)$ となり，M や l_k には依存しない[†8]．なお，2.1 節で述べた各既存手法のホップ数は，PIER と BF Skip Graph が $O(\log N)$，DST が $O(l_q + \log N)$ である．

SSA のホップ数について確認するため，シミュレーションプログラムを実装し，実験を行った．実験設定は以下のとおりである．

- 各物理ノードに，長さ l のランダムな数字からなるノードキーを 1 つずつ付与 (例：$l = 4$ の時，“0713” 等)．
- 付与したノードキーの中から，ランダムに 10 個のノードキーを選択．
- 各物理ノードが，上記で選択した 10 個のノードキーを検索文字列として，部分一致検索を実行 (即ち，各ノードが 10 回ずつ検索を実行)．
- 検索文字列にマッチするノードにクエリが到達するまでに要したホップ数を計測し，平均値を算出．
- 物理ノード数を 10，100，1,000，10,000 と変化させ，上記平均ホップ数の変化を記録．
- 以上の設定による実験を，3 種類のノードキー長 ($l = 4, 8, 16$) についてそれぞれ実施．

なお，SSA においては，各ノードはノードキーを複数保持可能であるが，各ノードキーは接尾辞に分解された上で扱われるため，ノードキーの数が直接性能に影響を与えるわけではなく，接尾辞の数が重要となる．そのため，本実験を含む以降の各実験においては，各ノードにノードキーを 1 つずつ付与し，必要に応じてノードキーの長さ (接尾辞数) を実験毎に設定している．

また本実験では，検索ホップ数の仮想ノード数及び物理ノード数に対する変化を調べることを目的としている．Skip Graph においては membership vector が一様に生成されることから，検索ホップ数はキーを構成する文字列 (文字の出現頻度や共起頻度の偏り) には影響を受けない．一方，提案手法においてノードキーの長さは仮想ノード数に直接影響するため，ここでは上記のようにノードキーをランダムな数字列とし，その長さを変化させる設定としている．それらの，各ノードに付与したノードキーの中からランダムに 10 個を選択して検索語とすることで，キー空間上の最小端から最大端までの範囲から検索対象を均一

†8 文献 [15] の 3.3 節において証明が為されている．

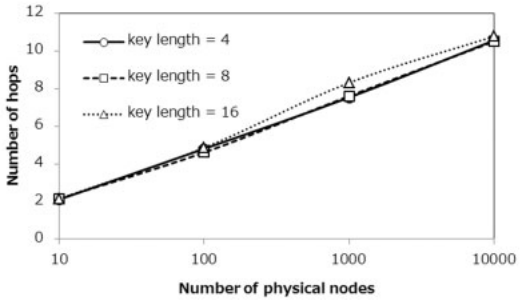

図 15　SSA の検索ホップ数

に選ぶ形となる．

図 15 は，横軸を物理ノード数，縦軸をホップ数として，実験結果をグラフにプロットしたものである．ノードキー長 l が大きくなるほど，Multi-key Skip Graph における仮想ノード数は増加するが，グラフでは l による違いがほとんど無く，物理ノード数にのみ依存していることがわかる．また，物理ノード数 N に対するホップ数の変化は対数オーダとなっていることが見て取れる．

4.2　経路表サイズ

次に，各ノードの経路表サイズについて考える．経路表の大きさは各ノードにおけるメモリ使用量に影響することに加え，ノードの挿入・離脱に伴う通信やリンクを維持するための通信等が増える要因となる．

2.1 節で述べた既存手法では，DHT や Skip Graph の経路表に加え，部分一致検索を実現するために各ノードが何らかの情報を保持することを義務付けられている．ここでは経路表の情報も含め，これらの，各ノードが保持を義務付けられる情報の単位をエントリと呼ぶこととする．各手法におけるエントリは以下のとおりである．

PIER　DHT の経路表の各行 (リンク)，及び他ノードから put されたポインタ

DST　DHT の経路表の各行，及び他ノードから put されたエッジ情報に含まれるポインタ

BF Skip Graph　Skip Graph の経路表の各行，及び Bloom filter

SSA　Skip Graph の経路表の各行

まず，各手法のエントリ数の比較を行う．PIER では，ノード毎の平均的なエントリ数は $O(\frac{Ml_k}{N}+\log N)$ となる．DST の場合，接尾辞木上の各エッジ毎に保持する情報の中にキーの集合が含まれており，このキー集合がポインタのようになっている．接尾辞木のエッジ数が最小の場合，即ち全接尾辞の接頭部が共通しないような場合には，平均的なエントリ数は $O(\frac{Ml_k}{N}+\log N)$ であり，エッジ数が最大の場合には $O(\frac{{Ml_k}^2}{N}+\log N)$ となる．BF Skip Graph については，平均的なエントリ数は $O(\log N)$ である．一方，SSA の場合，$\frac{Ml_k}{N}$ 個の仮想ノードそれぞれについて通常の Skip Graph 同様の経路表が必要になるため，エントリ数は平均的に $O(\frac{Ml_k}{N}\log N)$ となる．

前述のように，エントリ数はノード挿入等に伴う通信量にも影響する．ノード挿入時のコストについて考えてみると，PIER の場合，Chord を用いるとノードの挿入に伴うメッセージ数は平均的に $O(\log^2 N)$ であり，これに加えてポインタの put が発生するため，トータルでは $O(\log^2 N+\frac{Ml_k}{N}\log N)$ となる．DST では，ノードキーの各接尾辞を検索語とした検索処理を伴うため，$O(\log^2 N+\frac{Ml_k}{N}\log N+\frac{Ml_k^2}{N})$ となる．BF Skip Graph では，Skip Graph への挿入処理が完了した後，各ノードにおける Bloom filter の更新処理が為されなければ当該ノードのノードキーが検索可能とならない．従って，平均的に発生するメッセージ数は $O(N\log^2 N)$ となる．なお，2.2.5 項にて述べたように，更新処理において転送されるメッセージのサイズはフィルタサイズ b に依存する．SSA の場合は Multi-key Skip Graph に準ずるため，物理ノードの挿入に伴うメッセージ数は平均的に $O(\frac{Ml_k}{N}\log N)$ である．

なお，N や M の値は実際の用途により異なるが，例えば 5 章に述べているような応用例では，ノード数 N が数十万から数百万を超える程多く，またどのノードも 1 つあるいは数個程度のノードキーを持つような状況を想定しているため，M は N に比例して大きくなる．このような想定下では，特にノード挿入に伴うメッセージ数については，SSA に優位性があると言える．

続いて，ノード毎のエントリ数の偏りに着目する．

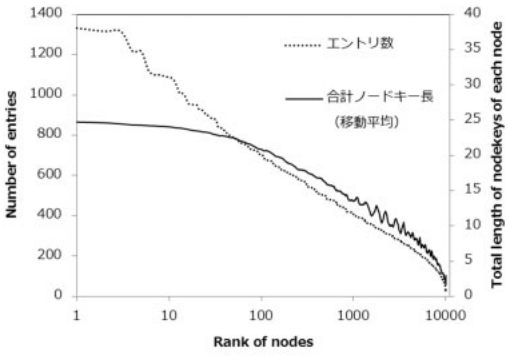

図 16 SSA におけるノード毎のエントリ数と保持ノードキー長

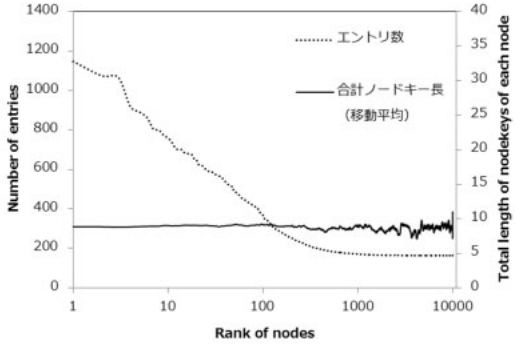

図 17 PIER におけるノード毎のエントリ数と保持ノードキー長

SSA の特徴として，エントリ数が，各ノードが登録を希望する情報量 (ノードキーの長さ) に応じた大きさとなるという性質がある．従って，例えば各ノードが同数の仮想ノードを挿入する場合は，ノード毎のエントリ数はほぼ等しくなる．一方，PIER や DST のような DHT ベースの手法では，ノード毎のエントリ数はキー (ノードキーの n-gram) の出現頻度偏りに大きな影響を受ける (2.2 節参照)．

この違いについて確認するため，SSA 及び PIER のシミュレーションプログラムを用意し，実験を実施した．なお，PIER における DHT アルゴリズムとしては Chord [10] を用いた．また，n-gram の出現頻度偏りはどのようなデータをノードキーとするかによって異なるが，4.1 節の実験のようにランダムな文字列をノードキーとすると出現頻度偏りによる影響が観測できないため，ここでは実際に用いられている単語集合の例として，ハッシュタグ集合 *tags* を用いることとした．実験設定は以下のとおりである．

- 物理ノード数は 10,000 とし，各物理ノードにノードキーとして *tags* に含まれるハッシュタグを 1 つずつ付与．
- 各物理ノードが持つノードキーの長さと，エントリ数を計測．

実験結果を図 16，17 に示す．グラフの横軸は，エントリ数の降順にソートした際の各ノードの順位であり，縦軸がエントリ数及びノードキー長である．なお，ノードキー長については視認性を考慮し，ウィンドウサイズを 200 とした移動平均によって平滑化している．また，エントリ数とノードキー長の相関係数を算出したところ，表 1 のようになった．

表 1 エントリ数とノードキー長の相関係数

	SSA	PIER
相関係数	0.985	0.0890

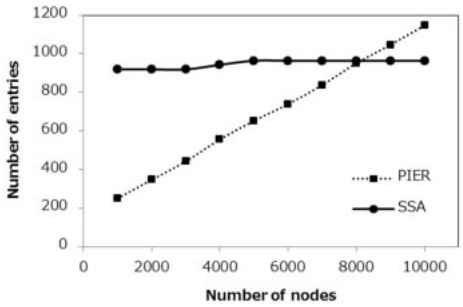

図 18 ノード数増加に対するエントリ数の変化

これらのグラフ及び表から，SSA においてはエントリ数がノードキー長に応じて変化しているのに対し，PIER ではノードキー長と無関係に n-gram の出現頻度偏りの影響を受けて偏っていることがわかる．

また，PIER では，P2P ネットワーク上に登録されるノードキー数の増加に伴ってエントリ数の偏りが拡大していく． 図 18 は，図 16，17 の実験と同様の設定で，エントリ数が大きいある 1 ノードのエント

リ数の変化を示したものである．10,000 ノードを順次参加させていく過程で，1,000 ノード参加時点でエントリ数が最も大きいノードに着目し，ノード数の増加に対するエントリ数の変化をプロットした．PIER ではノード数増加にほぼ比例する形でエントリ数が増大していることがわかる．

このように，PIER においては，特にノード数が数十万から数百万を超えるような大規模な P2P ネットワークを想定した場合，ごく一部のノードに極端に負荷が集中することになり，望ましくない．加えて，負荷が集中するノードはハッシュ値にもとづいて決まるため，意図的に特定のノードに負荷を集中させることも難しく，偶然低スペックな端末に負荷が集中してしまう事態も起こり得る．一方 SSA の場合，キーの出現頻度偏りの影響は受けず，各ノードのエントリ数は自身が保持するノードキー数とその長さに依存する．図 18 の実験設定では各ノードの保持するノードキーは変化しないため，エントリ数の変化も非常に小さなものとなっている．

4.3 検索に伴うメッセージ数

2.2 節に述べたように，PIER のような手法では 1 回の部分一致検索処理のために複数回の完全一致検索が必要となり，メッセージ数が多くなってしまう問題がある．また，ノード毎のメッセージの転送回数についても，n-gram の出現頻度偏りの影響を受けて大きく偏ってしまう．これに対し SSA では 1 回の検索処理で済むため，検索に伴い発生するメッセージ数を抑えることができる．

上記について確認するため，以下の実験設定により SSA と PIER それぞれについてシミュレーションを実施した．

- 物理ノード数は 10,000 とし，各物理ノードにノードキーとして *tags* に含まれるハッシュタグを 1 つずつ付与．
- *tags* 内の各ハッシュタグを検索文字列とし，検索開始ノードをランダムに選択して部分一致検索を実行 (即ち，10,000 回の検索処理を行う)．
- 各物理ノードのメッセージ転送回数を集計．

なお，ランダムな文字列を用いると n-gram の出現頻度偏りの影響が観測できないため，ノードキー及び検索語として *tags* を用いている．*tags* はある時間範囲内において Twitter 上で用いられたハッシュタグを重複を許して取得しているため，*tags* 内の各ハッシュタグを検索語としてランダムなノードから検索を行うことで，検索語の出現頻度も含め実データからシミュレートする形となっている．

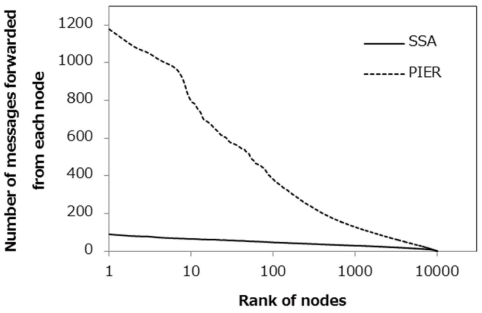

図 19　ノード毎のメッセージ転送回数

表 2　メッセージ転送回数の変動係数

	SSA	PIER
変動係数	0.564	1.263

実験の結果を，図 19 に示す．また，表 2 は物理ノード間のメッセージ転送回数に関する変動係数を算出したものである．図 19 及び表 2 から，SSA に対し PIER ではノード毎の転送回数が大きく偏っていることがわかる．またこのグラフからは，ネットワーク全体を流れる総メッセージ数にも大きな差があることが見て取れ，SSA においてネットワークにかかる負荷が大きく低減されていることがわかる．

また，SSA の特徴として，検索開始ノードの通信負荷が分散されることも挙げられる．これは，検索にマッチするノードキーが膨大に存在する場合でも，Skip Graph の範囲検索機構によりマルチキャストが行われるため，検索開始ノードから膨大な回数にわたりクエリを送信する必要がないことによる．

一方，PIER のような手法では，完全一致検索の組みわせにより検索にマッチする ID を特定後，実際に

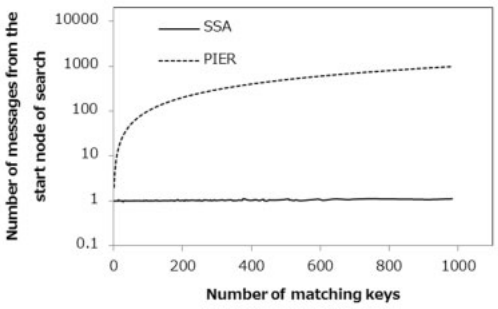

図 20 検索開始ノードから送信されるメッセージ数

当該 ID へクエリを送信する必要があり，その送信処理は全て検索開始ノードが行うことになる．

このことを確認するため，以下の実験設定により SSA と PIER それぞれについてシミュレーションを実施した．

- 物理ノード数は 10,000 とし，各物理ノードにノードキーとして *tags* に含まれるハッシュタグを 1 つずつ付与．
- *tags* に含まれる 2-gram を検索語として考えた時，マッチ数[†9]は 1 から 983 までの 232 種類であった．これら各マッチ数毎に，以下を 100 回繰り返す．
  - 当該マッチ数の 2-gram を検索語とする．(複数の 2-gram が該当する場合は毎回ランダムに選択する．)
  - 検索開始ノードをランダムに選んで検索を実行．
- マッチ数毎に，検索開始ノードのメッセージ送信回数の平均値を算出．

例えば，*tags* において 50 個のハッシュタグに含まれる 2-gram は “51”，“ft”，“AM” の 3 種類であったので，これらの中からランダムに選択したものを検索語として検索を実行することを 100 回繰り返し，検索開始ノードのメッセージ送信回数を計測する．そして，100 回の試行における送信回数の平均値を，マッチ数 50 における平均送信回数とする．

実験結果を図 20 に示した．この結果から，SSA ではマッチするノードの数にかかわらず検索開始ノードのメッセージ転送回数が約 1 回であるのに対し，PIER ではマッチ数に応じて検索開始ノードの転送回数が増加していることがわかる．

4.4 障害対策に関する特徴

構造化オーバレイでは，何らかの理由によりノードが離脱した場合，そのノードにリンクを張っていたノードの経路表が不完全なものとなり，正常なルーティングができなくなってしまう．そのため，ノードの急な離脱等に対する障害対策の検討が必須である．通常，各ノードは経路表として 2 つ以上のノードへのリンクを持つため，消失していないノードを経由して経路表を修復するのが一般的である．これは，SSA 及び 2.1 節で述べた既存手法のいずれにおいても求められる機構である．

DHT の場合，上記のような経路表修復とは別に，データの put に際して複製を配置する等の対策も施すことが多い．これは，DHT はデータの分散管理を実現する手法であり，ノードが離脱するとデータが失われてしまうためである．PIER や DST のような DHT ベースの手法では，ポインタを DHT 上に put する形になるため，正常な検索機能を維持するにはこういった対策が必要となる．

このような複製配置を行なった場合，被 put 数が複製数に応じて増加するため，ここまで議論してきたエントリ数等にも影響が及ぶことになる．即ち，PIER や DST の場合，実用的な観点からはより大きなエントリ数が必要となる．

4.5 検索機能への制約及び偽陽性の排除

SSA の場合，n-gram を用いる既存手法とは異なり，任意の文字列を指定した部分一致検索が可能である．また，検索結果についても，偽陽性が発生しない利点がある．

4.6 更新処理の有無

BF Skip Graph と SSA は共に Skip Graph を用いており，経路表を維持管理する仕組みについては共通であると言えるが，2.2.5 項で述べたように，BF

†9 検索語 (2-gram) を含むハッシュタグの数．

Skip Graph では各ノードにおける定期的な更新操作が必要である．そして，ノードの挿入・離脱等が検索結果に反映されるためには各ノードにおいて $O(\log N)$ 回の更新処理が為されるのを待つ必要がある．SSA の場合，このような更新処理は不要であるため，通信量を低減でき，ノードの挿入や離脱が検索結果に反映されるまでのタイムラグも少なくなっている．

4.7 より複雑な検索の実現

SSA における他の利点として，単純な部分一致検索以外にも実現可能な検索機能が存在することが挙げられる．

4.7.1 範囲を指定した部分一致検索

通常，ある範囲の文字列のいずれかを部分文字列として含むようなノードキーを検索することは困難である．例えば 2.1 節で述べた既存手法を用いて，「部分文字列として数字を含むもの」といった検索をしたい場合，当該範囲に含まれる文字列は文字列長を制限しなければ無限に存在しているため，範囲内の全文字列を列挙して検索処理することは困難である．

仮に列挙できたとしても，範囲内に含まれる文字列の数が多い場合，多くのクエリを投げることになり，検索コストが大きくなってしまう．上例の場合，2-gram であれば “00” から “99” までを調べれば 2 桁以上の数字を含むものは網羅できるが，100 回の部分一致検索処理が必要となる．

SSA では，部分文字列についての順序関係が Skip Graph 上で保たれているため，ある範囲の文字列のいずれかを部分文字列として含むようなノードキーを検索することが可能である．即ち，「部分文字列として数字を含むもの」といった検索を実現することができる．さらに，例えば「“Japan” の後に 2001 から 2010 までの数字が続くノードキー」等を検索することも可能である．

4.7.2 前方一致検索

ノードキーを接尾辞に分解する際，ノードキーの先頭に特別な記号を付与しておくことで，前方一致検索が可能となる．例えば，先頭記号として「^」を用いた場合，「^Japan」を検索文字列とすれば Japan から始まるノードキーを検索することができる．これにより，通常の Skip Graph と同様のノードキーに対する範囲を指定した検索も可能となる．

4.7.3 後方一致検索

前方一致検索と同様，ノードキーを接尾辞に分解する際に，ノードキーの末尾に特別な記号を付与しておくことで，後方一致検索が可能となる．但し，末尾記号の付与を行う場合は，3.4 節で述べた冗長な接尾辞の削減はできなくなる．

5 おわりに

本稿では，構造化オーバレイにおける部分一致検索手法として，Multi-key Skip Graph と接尾辞配列を応用した手法 SSA を提案した．従来の手法では，ノードによって負荷が大きく偏ってしまったり，検索結果に偽陽性が生じる等の問題があったが，SSA では $O(\log N)$ の検索ホップ数を達成しつつ，ノード負荷の偏りを解消し，通信負荷の低減や偽陽性の排除も実現している．また，範囲を指定した部分一致検索や前方一致検索，後方一致検索も可能である．

構造化オーバレイネットワークにおいて，リソースを柔軟に検索可能であることは，重要な機能上の要求である．特に，近年センサネットワークに向けたオーバレイネットワーク活用の研究が盛んである [13] が，センサネットワークでの利活用を考えた場合，性能の低いノードが膨大な数参加することになり，特定のノードに負荷が集中することは大きな問題となるため，従来手法における負荷の偏りを解消できる SSA には優位性があると考えられる．

提案手法の応用例としては，ノードの属性を文字列で表現し，特定の文字列を含む属性を持つノードへ情報を転送可能とするようなアプリケーションが考えられる．例えば，ユーザが保持する書籍のタイトルを属性として保持し，特定の文字列を含むタイトルの書籍を持つユーザに限定して情報を配信する等のシステムが実現可能である．

また，我々は構造化オーバレイを応用した P2P 型のコミュニケーションシステムを提案している [2]．このシステムは同一のウェブページを閲覧するユーザ同士がリアルタイムにコミュニケーションをとることを実現するものであり，各ユーザが閲覧中のウェ

ブページの URL が，本稿におけるノードキーのような役割を果たしている．即ち，ユーザは URL 毎にグルーピングされており，URL を指定することで，その URL を訪れているユーザ全員に対しメッセージを配信することができる．このシステムは DHT を用いているため，URL という完全一致指定が可能な情報をもとにユーザをグルーピングしているが，提案手法を応用することで，例えばウェブページのタイトルをノードキーとし，タイトルに含まれるキーワードでグルーピングする等，より柔軟なグループ管理が可能となると考えられる．

今後の課題としては，実環境での動作実験を行うことや，正規表現のようなより複雑な検索を実現することが挙げられる．また，本稿では検索にマッチするノードへのクエリ配信までを想定したが，クエリを受け取ったノードにどのような処理を行わせるかによってはさらに工夫が必要である．例えば，届いたクエリに対しデータを送り返すような用途では，マッチするノードが大量にあった場合，検索元ノードが結果を受信し切れない可能性がある．このような，特定の応用を想定した工夫について，より考察を深める必要があると考えている．

参 考 文 献

[1] Aspnes, J., Shah, G., Aspnes, J. and Shah, G.: Skip graphs, *ACM Transactions on Algorithms*, Vol. 3, No. 4(2007), pp. 1–25.

[2] Banno, R., Sato, H., Oyama, S. and Kurihara, M.: Uenew : A Cross-Browsing Communication System Based on Peer-to-Peer Networks, *Lecture Notes in Engineering and Computer Science*, Vol. 2188, Springer, 2011, pp. 697–701.

[3] Bloom, B. H.: Space/time trade-offs in hash coding with allowable errors, *Commun. ACM*, Vol. 13(1970), pp. 422–426.

[4] Castro, M., Druschel, P., Kermarrec, A. and Rowstron, A.: Scribe : A large-scale and decentralized application-level multicast infrastructure, *Selected Areas in Communications, IEEE Journal on*, Vol. 20, No. 8(2002), pp. 1489–1499.

[5] Harren, M., Hellerstein, J., Huebsch, R., Loo, B., Shenker, S. and Stoica, I.: Complex Queries in DHT-based Peer-to-Peer Networks, in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002, pp. 242–259.

[6] Manber, U. and Myers, G.: Suffix arrays: a new method for on-line string searches, in *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, 1990, pp. 319–327.

[7] Maymounkov, P. and Mazieres, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, in *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[8] Pugh: Skip lists : A probabilistic alternative to balanced trees, *Commun. ACM*, Vol. 33(1990), pp. 668–676.

[9] Rowstron, A. and Druschel, P.: Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems, in *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, 2001, pp. 329–350.

[10] Stoica, I., Morris, R., Karger, D., Kaashoek, M. and Balakrishnan, H.: Chord : A Scalable Peer-to-peer Lookup Service for Internet Applications, *ACM SIGCOMM Computer Communication Review*, Vol. 31, No. 4(2001), pp. 149–160.

[11] Weiner, P.: Linear pattern matching algorithms, in *Proceedings of the 14th Annual Symposium on Switching and Automata Theory* (*swat 1973*), 1973, pp. 1–11.

[12] Zhuge, H. and Feng, L.: Distributed Suffix Tree Overlay for Peer-to-Peer Search, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 20(2008), pp. 276–285.

[13] 吉田幹, 奥田剛, 寺西裕一, 春本要, 下條真司: マルチオーバレイと分散エージェントの機構を統合した P2P プラットフォーム PIAX(モバイルコンピューティング), 情報処理学会論文誌, Vol. 49, No. 1(2008-01-15), pp. 402–413.

[14] 岩本大記, 安倍広多, 石橋勇人, 松浦敏雄: P2P ネットワークにおける Skip Graph と Bloom Filter を用いた効率的な複数キーワード検索手法の提案, 情報処理学会研究報告, Vol. 2011-DPS-146, No. 28(2011), pp. 1–8.

[15] 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 拡張, 情報処理学会論文誌, Vol. 49, No. 9(2008-09-15), pp. 3223–3233.

坂 野 遼 平

2010 年北海道大学工学部情報エレクトロニクス学科卒業．2012 年同大学大学院情報科学研究科修士課程修了．同年日本電信電話株式会社入社．現在 NTT 未来ねっと研究所所属．

佐藤晴彦

2006 年北海道大学大学院情報科学研究科コンピュータサイエンス専攻修士課程修了．2008 年同大学大学院情報科学研究科複合情報学専攻博士後期課程修了．博士 (情報科学)．2009 年同大学院情報科学研究科助教．現在は関数型プログラムの形式的検証に関する研究に従事．2011 年電子情報通信学会論文賞受賞．

小山　聡

2002 年京都大学大学院情報学研究科社会情報学専攻博士後期課程修了．博士 (情報学)．2002–2007 年同大学院情報学研究科社会情報学専攻助手．2003–2004 年スタンフォード大学 Visiting Assistant Professor．2007–2009 年京都大学大学院情報学研究科社会情報学専攻助教．2009 年北海道大学大学院情報科学研究科複合情報学専攻准教授，現在に至る．2005 年度人工知能学会論文賞，2009 年度日本データベース学会上林奨励賞受賞．

栗原正仁

1980 年北海道大学大学院工学研究科情報工学専攻修士課程修了．同大学助手，講師，助教授，および北海道工業大学教授を経て，現在，北海道大学大学院情報科学研究科教授．2010 年より同研究科長．工学博士．ソフトウェア科学および人工知能の境界領域の研究に興味を持つ．1990 年情報処理学会創立 25 周年記念論文賞，2011 年電子情報通信学会論文賞受賞．情報処理学会，電子情報通信学会，人工知能学会各会員．