Vol.2010-DPS-144 No.9
2010/9/18

Range-Key Skip Graph による範囲検索可能な大規模分散キーバリューストアの実現

石　芳正[†1]　寺西裕一[†1,†2]　吉田幹[†3]
竹内亨[†2]　下條真司[†2]

オーバレイネットワーク Range-Key Skip Graph を用いることにより大規模環境においてもスケーラブルな範囲検索を実現可能とする分散キーバリューストアの構成法を提案する．Skip Graph 拡張の一つである Multi-Key Skip Graph を用いれば範囲検索可能な分散キーバリューストアを構成できるが，この方法では 1 つのキー毎に個別に経路表を持つため，キー数に比例して経路表のサイズが増加し，ノードの参加離脱時にオーバヘッドが大きくなる問題がある．そこで本研究では，ノードが担当するキーの集合を範囲キーとして扱い，Range-Key Skip Graph によりルーティングを行なうことで経路表を簡素化する方法を提案する．シミュレーション評価と実環境での動作確認を行ない，本提案により検索コストを Multi-Key Skip Graph と同等に保ちつつ，経路表を大幅に簡素化可能であることを確認した．

An Implementation of Large Scale Distributed Key-Value Store with Range Search Support based on Range-key Skip Graph

Yoshimasa Ishi,[†1] Yuuichi Teranishi,[†1,†2]
Mikio Yoshida,[†3] Susumu Takeuchi[†2]
and Shinji Shimojo[†2]

In this paper, a distributed range search supported key-value store construction method that by utilizing Range-key Skip Graph (RKSG) is proposed for applying a large-scale environment. Using Multi-key Skip Graph (MKSG) is a straightforward way to construct distributed key-value store with range search support. However, this method need to have a routing table for each key, therefore size of the routing table increases in proportion to the number of keys. When the enoumous number of data should be managed in the store, the management cost of routing table would be crucial. Therefore, a method to enable a routing table simplicity is proposed by handling set of keys in charge as a range key in RKSG. Our simulation results and evaluation in the large-scale environment show that the size of routing table can be greatly reduced, while the cost of search is almost the same with MKSG.

1. はじめに

今日のインターネットの普及により，世界中の人々を対象とした Amazon や Google，Facebook，Twitter といった事業者が現れ，数千万人から数億人に及ぶユーザにサービスを提供している．このれらの大規模サービスでは，サービスの成長にともない増加する保存データ量やトラヒックに追従するため，サービスを支える基幹プラットフォームには継続的な性能強化が必要であり，スケーラビリティが要求される．しかしながら，旧来の集中型システムで採られていた負荷の増加にあわせて高性能な機材に置き換えてゆくスケールアップ戦略では，コストに見合う性能が得られず，スケーラビリティの確保が困難になっている．このため，多数の安価な機材に処理を分散するスケールアウト戦略が採られる事例が増えてきている．ストレージシステムに注目すると，データの基本要素をキーと値のペアとしてこれら要素に対する操作を `GET` や `PUT` といった基本的なもののみとする代わりに，スケールアウトさせやすい仕組みを持つキーバリューストアが様々なサービスで採用されつつある．

スケールアウトを前提としたキーバリューストアとして，Amazon Dynamo[1] や Apache Cassandra[2]，ROMA[3]，kumofs[4] が挙げられる．これらのシステムでは，応答時間を抑える，もしくは保証するため，Consistent-hashing[5] に基づきキーバリューストアを構成するサーバ群の中から 1-hop（ no-hop，zero-hop とも呼ばれる）で目的のデータを持つ担当サーバに到達できる仕組みとなっている．これら Consisitent-hashing に基づくシステムは数百〜千台程度の規模を前提としたものが多い．また，Chord[6] や Kademlia[7] では，検索クエリがサーバ間で自律的に転送されることで目的のデータを持つ担当サーバに到達するマルチホップによる手法になっている．このマルチホップによる手法は Microsoft Azure[8] が採用しており，経路表を両方向に拡張した Chord を基盤としたストレージサービスを実現している．

1-hop に基づくシステムではクライアントもしくはゲートウェイサーバが直接担当サーバにアクセスできるため低遅延での応答が期待できるが，その一方でクライアントやゲートウェイサーバはストレージシステムを構成している全サーバへの経路情報を知っていなけれ

†1 大阪大学 大学院情報科学研究科
Graduate School of Information Science, Osaka University
†2 独立行政法人情報通信研究機構
National Institute of Information and Communication Technology
†3 株式会社ビービーアール
BBR Inc.

ばならない．一方，マルチホップに基づくシステムでは担当ノードまで順にホップしなければならないため，そのホップ数に応じて応答時間が変化し，1-hop に基づくシステムと比較して応答に時間を要する場合が生じるが，クライアントやゲートウェイサーバはストレージシステムを構成している一部のサーバへの経路情報を知っているだけでよい．担当サーバの発見に要するコストと経路情報の保守コストのトレードオフになるが，多数のサーバを利用した大規模ストレージシステムではサーバの故障発生頻度も増えるため，サーバ故障による影響についても考慮しなければならない．

サーバ故障に対する耐性を確保するには，故障サーバを経路情報から取り除くなどの回復処理が必要になるが，1-hop に基づくシステムでは全てのクライアントやゲートウェイサーバが故障サーバへの経路情報を持つため，それら全てに対して回復処理が必要になる．一方，マルチホップに基づくシステムでは，あるサーバへの経路情報を持つサーバは一部のサーバに限られることが多く，回復処理はそれら一部のサーバが実施するだけでよい．このため，回復処理に要するコストは 1-hop に基づくシステムより抑制できると期待できる．

また，キーバリューストアの機能面に注目すると，Twitter では Twitter クライアントが Web API により新しいタイムラインを取得する際に「ある ID の発言より新しい発言の取得」という操作が頻繁に生じており，この操作には範囲検索が有効であると考えられる．しかしながら，Dynamo や ROMA，kumofs では与えられたキーに対してハッシュ処理を行ってからデータを格納するため，元のキーの順序関係が保存されず範囲検索をサポートしていない．逆に範囲検索が可能であれば，指定 ID 以降の発言データを 1 度の検索要求でまとめて取得することができ，ストレージシステムへの問い合わせ頻度を削減できる．この他にも「ある期間のイベントログ情報」や「位置に紐付けられた写真情報」といった範囲検索が必要となる場面があるが，ハッシュ処理を用いたキーバリューストアだけでは実現が難しく，別の仕組みで範囲検索機能を実現するためにシステムが複雑になってしまう．

今後 Twitter などのサービスがより多くのユーザを獲得して成長することを考えると，さらに多くのサーバを統合可能とし，範囲検索も行えるキーバリューストアへの要求は高まると思われる．この要求を満たすには，サーバ同士がマルチホップに基づき連携し，範囲検索もその上で実現しなければならない．この 2 要件を満たす仕組みには Skip Graph[9] やこれを拡張した Multi-Key Skip Graph[10]，Range-Key Skip Graph[11] がある．本研究では，これらの内 Range-Key Skip Graph を用いて，数十万〜数百万台規模での動作を可能とし，さらに範囲検索を行えるキーバリューストアを提案する．

2. 範囲検索可能なキーバリューストア

2.1 分散キーバリューストアモデル

分散キーバリューストアにおいて扱うデータは，キー（ Key ）と値（ Value ）であり，ユーザ（クライアント）がキーを指定して，対応する値の入出力を行なう．値は，複数のノード（ Node ）に分散して保存される．

図 1 は分散キーバリューストアのモデルを示している．Storage Key Space は，分散キーバリューストアを構成するノード群が取り扱うキー（ Storage Key ）の値域を表している．分散キーバリューストアでは，各ノードが Storage Key Space 上の特定の領域を担当し，値の分散管理を行う．すなわち，Storage Key Space からノードへのマッピング g が必要となる．

User Key Space は，キーバリューストアにおいてユーザが指定するキー（ User Key ）の値域を表わすキー空間である．User Key Space と Storage Key Space は必ずしも一致しない．すなわち，User Key Space から Storage Key Space へのマッピング f が必要となる．

通常，分散キーバリューストアでは，ノードの出入りがあっても値を返却できるよう，単一の User Key に対応する値を複数のノードに分散して持たせる．よって，g は，通常 Storage Key から複数のノードへのマッピングとなる．

Chord，Kademlia 等の代表的な分散ハッシュテーブルをこのモデルに当てはめると，User Key Space がキー，f がハッシュ関数となり，Storage Key Space がハッシュ値の空間に対応する．g には，ノードの ID に f と同一のハッシュ関数を適用して区切られた領域と当該 ID を持つノードのマッピングが対応する．各ノードにおいて，ノードの割り当て領域外のハッシュ値を探し出すためにルーティングアルゴリズムが存在し，Chord では successor list, finger table によるマルチホップ検索が用いられる．

従来の分散キーバリューストアが持つ入出力インタフェースは，User Key に対応する値を格納する `PUT`，単一の User Key に対応する値を取得する `GET` が基本である．`PUT`（Ku，V）を実行した場合，$g(f(Ku))$ により得られるノード集合に，Ku, V を対応付けて格納する．`GET`（Ku，V）を実行すると，$g(f(Ku))$ により得られるノード集合から，Ku に対応付けられた V を取得する．

2.2 分散キーバリューストアにおける範囲検索

ここで，分散キーバリューストアの入出力インタフェースとして，指定範囲内にあるキー

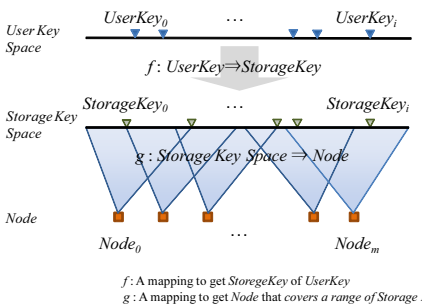

図 1 分散キーバリューストアモデル
Fig. 1 A model of distributed key-value store

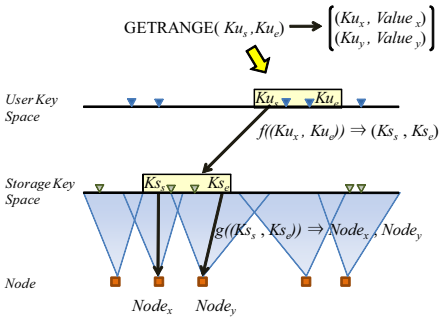

図 2 分散キーバリューストアにおける範囲検索
Fig. 2 Range search of distributed key-value store

と値のペアを全て取得する範囲検索 GETRANGE(Ku_s, Ku_e) インタフェースを追加することを考える．

f がハッシュ関数のように順序が保存されないマッピングである場合，範囲 $[Ku_s, Ku_e]$ に対応する Storage Key Space 上の範囲を得ることができない．よって範囲検索を行うには，f は順序が保存されるマッピングである必要がある．また，g としては，$[f(Ku_s), f(Ku_e)]$ に対応するノードを全て取得する必要があり，その数が膨大とはならないマッピングであることが望ましい．

範囲検索 GETRANGE(Ku_s, Ku_e) を実行した場合，$[f(Ku_s), f(Ku_e)]$ より得られる Strage Key Space 上のキー範囲 $[Ks_s, Ks_e]$ がマッピングされたノード集合（$Node_x, Node_y$）から $[Ku_s, Ku_e]$ に含まれるキーと値のペアを全て取得する（**図 2**）．

3. Multi-Key Skip Graph による分散キーバリューストア

前章の g として，$[f(Ku_s), f(Ku_e)]$ に対応するノードを全て取得することができるルーティング方式として，Multi-Key Skip Graph がある．以下では，Multi-Key Skip Graph について概説する．

3.1 Multi-Key Skip Graph

Multi-Key Skip Graph は Skip Graph に拡張を行ったオーバレーネットワークであり，単一のノード上に複数のキーを保持可能としたオーバレイネットワークである．ここではまず，基本アルゴリズムである Skip Graph を用いて説明する．

Skip Graph は Skip List[12] を分散環境に適用し，構造化オーバレイネットワークとしたものであり，分散環境下で目的とするキーを持つノードを発見できる．分散ハッシュテーブルとは異なり，元のキーをハッシュ処理せずにオーバレイネットワーク上で扱い，範囲検索を可能としている．このキーは全順序関係を定義可能であればどのような要素であっても利用可能である．

Skip Graph では次の 2 種類の検索をサポートしている．前者は分散ハッシュテーブルと同様の完全一致検索，後者が範囲検索になる．

- ある値を指定して，その値を持つキーを検索する値対値検索
- ある範囲を指定して，その範囲に含まれるキーを検索する範囲対値検索

すなわち，検索クエリを Q としたとき，以下の 2 条件のいずれかを満たす Skip Graph 上のキーを K_i が発見される（Qs は検索クエリ Q が持つ範囲の最小値，Qe は最大値を表す）．

- $K_i = Q$
- $Qs \leq K_i \leq Qe$

図 3 に Skip Graph で実現される検索例を示す．図中下部は Skip Graph を構成する各サーバを示しており，それぞれのサーバがキーを持ちそれらがキー空間上に配置されている様子を示している．この例ではサーバ A がキー 18，サーバ B がキー 5，サーバ C がキー 22 を持ち，それぞれ Skip Graph の 1 ノードとして動作している．任意のサーバよりキー 5 を検索した場合，該当するキーを持つサーバ B が発見される．同様に 16 から 25 の範囲指定により検索した場合，範囲内に含まれるキー 18 とキー 22 を持つサーバ A とサーバ C が発見

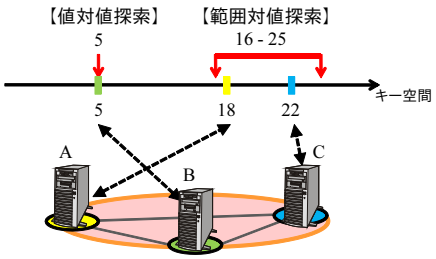

図 3　Skip Graph における検索例
Fig. 3　An example of searching keys on Skip Graph

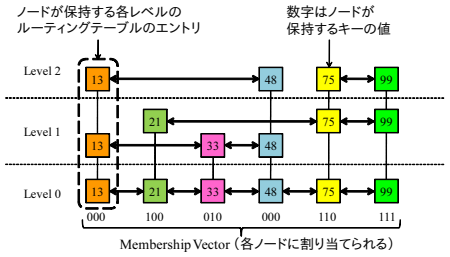

図 4　Skip Graph の構造
Fig. 4　The routing structure of Skip Graph

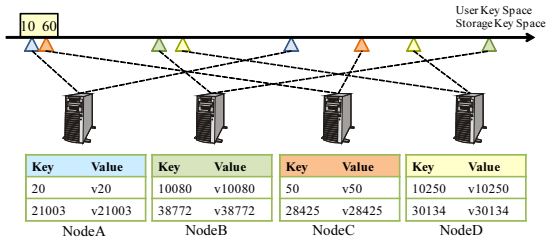

図 5　Multi-Key Skip Graph によるキーバリューストア
Fig. 5　Multi-key Skip Graph based key-value store

される．

続いて Skip Graph オーバレイの構造について概説する．オーバレイの構造は **図 4** となっており，図中の縦に並んだ正方形の組がノード上に保持されるキーを表し，中の数値がキーの値を表している．個々の正方形はノードが持つ経路表の 1 エントリを表し，正方形間を横に結んだ矢印がキー間の経路を表している．経路表の各エントリはそれぞれ図中左側に示したレベルに属し，レベルごとに隣接するキーの値とそのキーを保持しているノードへの経路情報を持つ．

各キーの下部に示した 2 進数値は membership vector であり，各レベルにおいて隣接とするキーを決定するために使用される．具体的には，レベル i において membership vector の接頭 i 桁が等しいキーが隣接するキーとなり，各レベルにおいて接頭 i 桁が等しいキーのエントリ群により構成されたリストが接頭 i 桁の種類だけ存在することになる．

この Skip Graph の 1 ノードが持つ経路情報数を考えると，Skip Graph を構成するノード数が十分に多くその数を m とした場合，レベルの高さは $log(m)$ となる．この各レベルにおいて両隣のキーへの経路情報を持つため，1 ノードあたりの経路情報数は $2 \times log(m)$ となる．

Multi-Key Skip Graph は，この Skip Graph を拡張し，1 ノード上に複数のキーを保持可能とした上で，検索時におけるメッセージの多重処理を防ぐマルチレンジフォワーディングにより効率のよい検索を実現している．

Multi-Key Skip Graph では，キー毎に経路表は持つため，ノードが持つキー数を i とすると 1 ノードが持つ経路情報数は $2 \times log(m) \times i$ となる．

3.2　分散キーバリューストアの実現方法と問題点

Multi-Key Skip Graph では，キーの値域に制限がないため，User Key Space と Storage Key Space は同一キー空間とすることができる．したがって，f は同一のキーへのマッピングとする．また， Multi-Key Skip Graph においては，ノードが持つキーに対応してルーティングが行なわれるため，g にも制約がなく，毎回対応する値が変わるランダムなマッピングであっても構わない．ただし，ランダムなマッピングではキーに対応する値を格納するノード数が多くなってしまうと考えられるため，ある程度順序が保存され，ノードあたりの担当キー数が分散するマッピングを行なうことが望ましいといえる．

すなわち， Multi-Key Skip Graph による分散キーバリューストアは以下の動作をすることとなる．

`PUT`$(Ku,\ V)$ では，マッピング g により得られるノードすべてが Multi-Key Skip Graph 上に Ku を加え，Ku と V を保存する．g は，必要数の担当ノードが得られるマッピングであればよく，制限はない．

`GET`(Ku) では，Ku のキーを持つノードへのルーティングを行ない，当該ノードから Ku に対応する V を取得する．

`GETRANGE`$(Ku_s,\ Ku_e)$ では，$[Ku_s, Ku_e]$ の範囲のキーを持つノードへのルーティングを行ない，当該ノードから $[Ku_s, Ku_e]$ にあるキーと値のペアを取得する．

図 5 は Multi-Key Skip Graph によるキーバリューストアの構成例を示している．この状態で `GETRANGE`$(10,\ 60)$ を実行すると，範囲内のキーを持つ $NodeA$，$NodeC$ が検索され，$[10, 60]$ の範囲内にある $(20, v20), (50, v50)$ が取得できる．

3.1 節で述べたとおり， Multi-Key Skip Graph では，各ノードが保持するキー毎に経路表

が必要となる．したがって，格納されたキー数に比例して各ノードが管理すべき経路表のサイズが大きくなる．Multi-Key Skip Graph では，ノードの出入りに際して関係するノードの経路表を更新する必要があるため，経路表のサイズが大きくなると，ノードの出入り時のオーバヘッドが大きくなってしまう．

4. Range-Key Skip Graph による分散キーバリューストア

本研究では，格納されたキー数に比例して各ノードが管理すべき経路表のサイズが大きくなる問題に対処するため，Range-Key Skip Graph により分散キーバリューストアを実現する方法を提案する．Range-Key Skip Graph は， Multi-Key Skip Graph を拡張し，範囲対範囲の検索を可能とする構造化オーバレイネットワークである．以下では，まず Range-Key Skip Graph について概説する．

4.1 Range-Key Skip Graph

Range-Key Skip Graph は Multi-Key Skip Graph を基に，上限値と下限値で指定される範囲を持つキー，レンジキーを扱うために拡張されたオーバレイネットワークである．Range-Key Skip Graph ではレンジキーを保持可能となったことにより Skip Graph がサポートしていた 2 種類の検索に加えて範囲も対象となるため，新たに次の 2 種類の検索が可能となっている．

- ある値を指定して，その値を含むレンジキーを検索する値対範囲検索
- ある範囲を指定して，その範囲と重複部分を持つレンジキーを検索する範囲対範囲検索

検索クエリを Q としたとき，以下の 3 条件のいずれかを満たす Range-Key Skip Graph 上のレンジキー RK_i が発見される（Qs は検索クエリ Q が持つ範囲の最小値，Qe は最大値を表す．RK_is はレンジキー RK_i の最小値，RK_ie は最大値を表す）．

- $RK_is \leq Qe \leq RK_is$
- $RK_is \leq Qs \leq RK_ie$
- $Qs \leq RK_is$ かつ $RK_ie \leq Qe$

図 6 は Range-Key Skip Graph で実現される検索例を示している．サーバ A がキー 18，サーバ B がレンジキー 3 ～ 8，サーバ C がレンジキー 22 ～ 23 を持ち，Range-Key Skip Graph の 1 ノードとして動作している．キー 5 を検索した場合，5 を含むレンジキーを持つサーバ B が発見される．16 から 25 の範囲検索を行った場合，検索範囲と重複部分を持つキー 18 とレンジキー 22 ～ 23 を持つサーバ A とサーバ C が発見される．

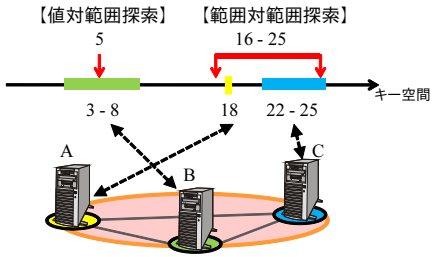

図 6 Range-Key Skip Graph における検索例
Fig. 6 An example of searching keys on Range-key Skip Graph

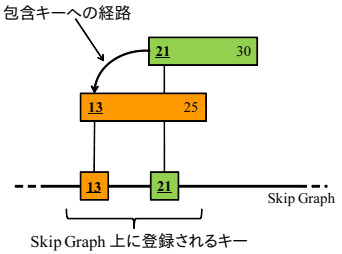

図 7 Range-Key Skip Graph の構造
Fig. 7 The routing structure of Range-key Skip Graph

続いて，Range-Key Skip Graph オーバレイの構造を概説する．Range-Key Skip Graph オーバレイの構造は **図 7** に示すようになっている．Range-Key Skip Graph の基本構造は Skip Graph と同じ構造を持ち，レンジキーが持つ範囲の最小値が Skip Graph 上に登録されたキーと同様の働きをし，経路表についても同様の構造となる．これに加えて，各レンジキーは自身の範囲の最小値を含む範囲を持つ他のレンジキー（包含キーと呼ぶ）への経路情報を持つ．また，Skip Graph 部は Multi-Key Skip Graph と同様の拡張を施すことができる．すなわち，Range-Key Skip Graph の経路表は，Multi-Key Skip Graph の経路表に包含キーへの経路情報を加えたものになる．ノードが持つレンジキー数を i，各レンジキーが持つ包含キーのリンク数の平均を k とすると Range-Key Skip Graph 1 ノードが持つ経路情報数は $(2 \times log(m) + k) \times i$ となる．

4.2 Range-Key Skip Graph による分散キーバリューストア

Range-Key Skip Graph の検索動作は見方を変えると レンジキーを持つノードにその範囲に重複する範囲条件を持つ検索クエリを送り届ける仕組みと見ることができる．すなわち，ノードがレンジキーを持つことはそのノードにレンジキーに相当する Storage Key Space の範囲をマッピングすることに相当する．Range-Key Skip Graph においては，ノードが持つレンジキーに応じてルーティングが行われるため，g には制約はなく，Storage Key Space を任意に分割すればよい．

また，Range-Key Skip Graph ではレンジキーの値域に制限がないため，User Key Space の値域を対応した Storage Key Space とすることで f は同一キーへのマッピングとすることができる．

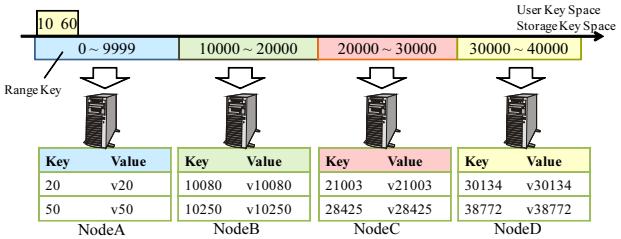

図 8　Range-Key Skip Graph によるキーバリューストア
Fig. 8　Range-key Skip Graph based key-value store

すなわち，Range-Key Skip Graph による分散キーバリューストアは以下の動作で実現される．

PUT（Ku，V）では保持するレンジキーの範囲に Ku が含まれるノードにルーティングを行い，V を Ku に対応づけて当該ノードのローカルストアに保存する．

GET（Ku）では保持するレンジキーの範囲に Ku が含まれるノードにルーティングを行い，当該ノードのローカルストアから Ku に対応する V を取得する．

GETRANGE（Ku_s，Ku_e）では $[Ku_s, Ku_e]$ を検索クエリの条件とすることで $[Ku_s, Ku_e]$ に重複するレンジキーを持つノードにルーティングし，当該ノードから $[Ku_s, Ku_e]$ に含まれるキーと値のペアを取得する．

図 8 は Range-Key Skip Graph によりキーバリューストアの構成例を示している．図では，正数空間を User Key Space，Storage Key Space として幅 10000 ずつに区切り，その範囲に相当するレンジキーを各ノードに対応付けている．この状態で GETRANGE（10，60）を実行すると，範囲に重複するレンジキー $[0, 99999]$ が検索され，レンジキーを持つ $NodeA$ より $[10, 60]$ の範囲内にある $(20, v20), (50, v50)$ が取得できる．

このように Range-Key Skip Graph をキーバリューストアに用いる場合，各ノードには Storage Key Space を割り当てるために事前にレンジキーを持たせることができる．この場合，各レンジキーの順序は既知となるため，レンジキーを持つノードの順序も既知となる．この順序情報を利用し，ノードの順序に即した membership vector を各ノードに与えることでランダム値を membership vector に用いた場合に比較して少ないホップ数で検索クエリを転送できる経路表を構築できる．具体的には，最小のレンジキーを持つノードを 0 番とし，以降レンジキーの順に従い 1 番，2 番と連番を割り振る．この番号をビット反転させた値を各ノードの membership vector とする．3bit の membership vector での例を挙げると，順に 000，100，010，110，001，101，011，111 という membership vector を持つことになる．経路表は membership vector の上位ビットの合致長に応じて構築されるため，各レベルでバランスする経路表となる．

5. キーバリューストアの実装と評価

5.1 キーバリューストアの実装

図 9 にサーバマシン上で動作させるキーバリューストアサーバの実装を示した．Key-Value RPC Interface は先に示した API を実装した RPC モジュールで，これを介してクライアントと Key-Value Connector を結びつける．Local Storage はサーバが持つレンジキーの範囲内のキーと値のペアを保持する．R-Key SG Overlay Interface は Range-Key Skip Graph を介した別のサーバとの通信を司る．Key-Value Connector は Local Storage，R-Key SG Overlay Interface，Key-Value RPC Interface を結びつけ，各モジュールからの要求や応答の処理を行う．Range-Key Skip Graph は PIAX[13] の実装を利用した．

クライアントが発した PUT や GET，GETRANGE 要求は RPC，すなわち Key-Value RPC Interface を介して Key-Value Connector が受け付ける．この要求が自サーバの担当範囲内への要求であった場合は，Local Storage に対して入出力処理を行い，その結果をクライアントに返す．自サーバの担当範囲外であった場合は，R-Key SG Overlay Interface を介して Range-Key Skip Graph で結ばれた担当サーバを検索し，要求を通知する．担当サーバより応答が得られるとその結果をクライアントに返す．他サーバより R-Key SG Overlay Interface を介して要求が届いた場合，その要求は Key-Value Connector に受け渡され，要求に応じて Local Storage に入出力処理を行う．Local Storage から得られた結果を Range-Key Skip Graph を介して要求元に送り返す．

5.2 評　　価

分散キーバリューストアについて Multi-Key Skip Graph による実装と Range-Key Skip Graph による実装の比較を行った．スケーラビリティに繋がる評価として 1 ノードが管理する平均経路数，GET，GETRANGE 時の平均最大ホップ長を計測した．

キーバリューストアは 100 ノードで構成し，キーを 0 から順に 1 ずつ増加させ 100 万個のキーと値を登録した．その際，10 万個登録するごとに各評価値を計測する．Range-Key Skip Graph による実装では，1 万幅のレンジキーを順に各ノードに持たせ，キー空間を各

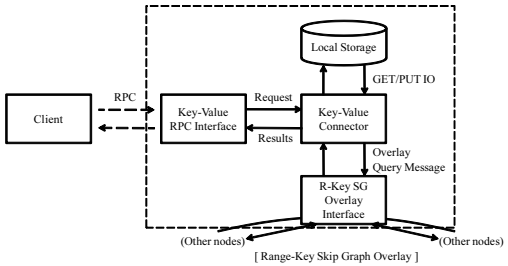

図 9 キーバリューストアサーバの構成

ノードに担当させている．Multi-Key Skip Graph による実装では，最もシンプルな方法としてクライアントから PUT 要求を受け付けたノードがそのキーを保持する仕様とした．また，PUT するノードの選択法として 1 万ずつ PUT 先ノードを順に切り替える Multi-Key(Ordered) 手法と，ランダムに選択する Multi-Key(Random) 手法の 2 手法を用いた．Range-Key Skip Graph，Multi-Key Skip Graph 共に各ノードの membership vector は 4.2 節で示した手法で割り当てている．

まず，ノードあたりの平均経路数を **図 10** に示す．Multi-Key(Ordered)，Multi-Key(Random) では Multi-Key Skip Graph においてキー毎に経路表が必要になることから，キー数に比例して経路数が増えている．Range-Key Skip Graph ではノードが保持しているキー数によらず経路表は 1 つであるため，Multi-Key Skip Graph による実装と比較して非常に少ない経路数（ 13.5 ）となっている．

図 11 と**図 12** はそれぞれ各手法における GET と GETRANGE 時の平均最大ホップ長を表している．Multi-Key Skip Graph では GETRANGE した場合，一般に複数のノードが検索されるため，各ノードへの検索経路のうち最大のホップ数を要したものを最大ホップ長としている．最大ホップ長の計測は，ランダムにノードを選び，そのノードより操作を実行するという処理を GET と GETRANGE それぞれ 1000 回行い，その結果の平均を求めている．取得対象はキーが登録されている範囲からランダムに選択し， GETRANGE では範囲幅を 300 を上限としたランダムな範囲とした．

両操作ともに，Multi-Key(Random) のホップ数が明らかに大きくなっている．これは，4.2 節による順序を考慮した membership vector を設定したにもかかわらずキー登録先ノードをランダムとしたため，逆効果になりホップ数が増えたと考えられる．特に GETRANGE では，指定された範囲に含まれるキーは複数のノードに分散することとなり，その中で最も長い経路が最大ホップ長であることから他の手法と比較して約 4 倍のホップ数を要している．

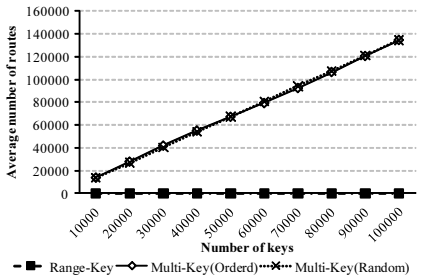

図 10 ノードあたりの平均経路数
Fig. 10 The average number of routes per node

これに対して Range-Key と Multi-Key(Ordered) では，キーが特定のノードに集約された状態になるため，集約されたキー数と範囲検索の幅を考慮すると複数の経路に分かれることは希である．Multi-Key(Ordered) ではマルチレンジフォワーディングにより Range-Key とほぼ同等のホップ数となっている．キー数が少ない時点では Range-Key より多くのホップ数を要しているが，これはキーが登録されていないノードからのクエリ発行が多いためであり，キーの登録が進むに従いキーが登録されているノードからのクエリ発行が増え，その分ホップ数が減少することによる．最終的に Range-Key よりも Multi-Key(Ordered) のホップ数が少なくなるが，これは Range-Key と Multi-Key のルーティング方法の差異によるものと思われる．

以上の評価より Range-Key Skip Graph を用いた場合， Multi-Key Skip Graph を用いた場合とほぼ同等か，キー配置方法によっては大幅に少ないホップ数で GET，GETRANGE を行えることがわかる．その一方で，保持すべき経路情報数はより少なくて済むため，ノードの出入りによる経路表修正コストを削減できることが期待できる．

6. 大規模環境での動作確認

本研究で提案した Range-Key Skip Graph によるキーバリューストアの大規模環境における動作を確認するため，情報通信研究機構北陸リサーチセンターが運用する大規模ネットワーク実験用 PC クラスタである StarBED を用いて 1024 × 1024 ノードからなる大規模分散キーバリューストアを構築した．物理的には StarBED のサーバ 1 台につき 1 万ノードを

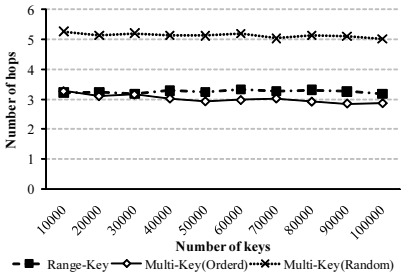

図 11 GET 時の平均最大ホップ長
Fig. 11 The average number of maximum hops at GET

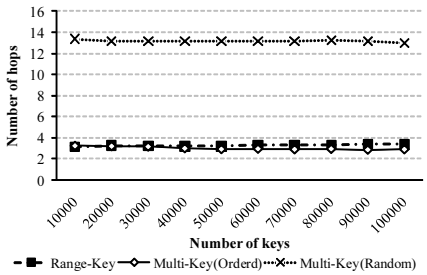

図 12 GETRANGE 時の平均最大ホップ長
Fig. 12 The average number of maximum hops at GETRANGE

動作させ 105 台のサーバを用いる形式としている．

動作確認のため，位置情報を持つデータの緯度経度をキーとして格納し，格納されたデータからユーザが指定する範囲のデータを取り出して地図上に可視化するアプリケーションを作成した．このアプリケーションを用いて LiveE![⋆1] から気象観測値と iPhone アプリから得られる移動履歴をそれぞれ定期的に StarBED 上のキーバリューストアに PUT し，ダミーデータを含めて全体で約 1000 億個のデータを格納できることを確認した．加えて，GETRANGE によりユーザが指定する範囲内のデータが取得できることを確認し，大規模なデータを取り扱い可能であることを実証した．[⋆2]

7. ま と め

本研究では，構造化オーバレイネットワークである Range-Key Skip Graph による分散キーバリューストアを構築した．従来の分散キーバリューストアでは一般的ではなかった範囲指定によるデータ取得において，Range-Key Skip Graph を用いることで各ノードが持つ経路情報数を抑制し，数十万～数百万台へのスケールアウトの目処をたてた．

今後の課題として，現時点ではレプリカを考慮しておらず，ノード障害時には担当範囲への入出力が行えなくなるため，レプリカの配置について検討する必要がある．また，負荷分散についても考慮していないため，アクセス負荷とストレージ負荷の両面において負荷の集中が生じた際の分散手法の検討が必要である．

謝辞 本研究の一部は，総務省委託研究「ユビキタスサービスプラットフォーム技術の研究開発」による成果である．また，大規模環境下での動作検証に独立行政法人情報通信研究機構北陸リサーチセンターが運用する StarBED ，及び同機構運用の JGN2plus を利用した（JGN2P-A20089 「PIAX に基づく広域オーバレイネットワークを用いたユビキタスサービスプラットフォームの検証」）．利用に際し各々の運用チームの皆様には多大なるご協力を頂いた．ここに記して謝意を表す．

⋆1 LiveE! Project: http://www.live-e.org/
⋆2 このシステムは 2009 年 10 月 6 日 から同 10 日にかけて開催された CEATEC 2009 にて一般展示を行った．

参 考 文 献

1) DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P. and Vogels, W.: Dynamo: amazon's highly available key-value store, *SIGOPS Oper. Syst. Rev.*, Vol.41, No.6, pp.205–220 (2007).
2) The Apache Cassandra Project. available at http://cassandra.apache.org/.
3) Rakuten ROMA. available at http://code.google.com/p/roma-prj/.
4) The Kumofs Project. available at http://kumofs.sourceforge.net/.
5) Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M. and Lewin, D.: Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, New York, NY, USA, ACM, pp.654–663 (1997).
6) Morris, R., Karger, D., Kaashoek, F. and Balakrishnan, H.: Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications, *ACM SIGCOMM 2001*, San Diego, CA (2001).
7) Maymounkov, P. and Mazieres, D.: Kademlia: A Peer-to-Peer Information System Based on the XOR Metric, *the 1st International Workshop on Peer-to-Peer System(IPTPS'02)*, p.263 (2002).
8) Windows Azure Platform. available at http://www.microsoft.com/windowsazure/.
9) Aspnes, J. and Shah, G.: Skip Graphs, *ACM Transactions on Algorithms*, Vol.3, No.4, p.37 (2007).
10) 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, pp.3223–3233 (2008).
11) 石 芳正, 寺西 裕一, 吉田 幹, 下條 真司, 西尾 章治郎：範囲をキーとして保持可能とする Skip Graph 拡張の提案, 情報処理学会研究報告 (2009-DPS-139),Vol.2009, pp.1–7 (2009).
12) Pugh, W.: Skip Lists: A Probabilistic Alternative to Balanced Trees, *Workshop on Algorithms and Data Structures*, pp.437–449 (1989).
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エントの機構を統合した P2P プラットフォーム PIAX , 情報処理学会論文誌, Vol.49, No.1, pp. 402–413 (2008).