M2M データのための
インデキシング技術の研究

荒川　豊

電気通信大学大学院情報理工学研究科

博士（工学）の学位申請論文

2014 年 12 月

M2M データのための
インデキシング技術の研究

博士論文審査委員会

主査　市川　晴久　教授

委員　太田　和夫　教授

委員　吉浦　裕　　教授

委員　西野　哲朗　教授

委員　羽田　陽一　教授

Study on Indexing Technology for M2M Data

Yutaka Arakawa

Abstract

In recent years, M2M (Machine-to-Machine) applications have become more diverse and widespread because of a reduction in price and growth in the capacity of network infrastructures, and because of the miniaturization of communication devices. M2M applications are those applications that use communication between machines. In addition, there is a growing recognition of the need for horizontal-integrated platforms that allow M2M applications to share common functions, devices, and data. Horizontal-integrated platforms allow a reduction in the cost and time required to implement such M2M applications.

In order to accelerate the spread of M2M applications, this study aims to realize the data search function that is one of the basic functions in horizontal-integrated platforms. Indexing technology is necessary for the efficient search of big data. In addition, appropriate indexes vary according to data and queries. Therefore, the goal of this study is to establish new indexing technologies suited to M2M data, and query of M2M applications. The search function requires rapidity and scalability for both insert process and search process as significant amount of data are accumulated; furthermore, the search function is required to execute several search processes concurrently. The more frequently a search condition is used, the more efficiently the search condition should be processed. However, it is impossible to predict the frequency of each search condition in advance because M2M applications can use various search queries; therefore, the frequency can change dynamically. It is difficult to determine which and how many search conditions should be processed efficiently. Therefore, the search function requires load adaptability to allow such function to be flexible with changes in the input data and search queries.

In this study, search requests from M2M applications are listed and classified first. As a result, we find that there are five search patterns and that one of these patterns is used most frequently. This pattern is a two-dimensional search query that uses the search condition that specifies two dimensions (attributes) of “ID” of the device and measurement “time” by the device. However, from the aspect of indexing technologies,

we find that the search pattern classified as “other multidimensional search pattern,” which includes various multidimensional search queries on various M2M data, cannot be processed with the aforementioned requirements, that is, rapidity, scalability, and load adaptability through conventional indexing technologies. M2M data includes many dimensions, such as “ID,” “time,” and “location.” The multidimensional search query is the query that specifies search conditions regarding multiple dimensions. The “other multidimensional search pattern” can be divided into multidimensional exact match search and multidimensional range search. The former is the search that specifies several attributes of things, such as “‘red’ ‘bag’ made in ‘2014’ by ‘X Company’.” The latter is the search that specifies several dimension value-ranges, such as time, location, sensor value, and so on, for example, “list of trucks whose truckload is ‘equal to or lower than 0.5 ton’ in the ‘Kanto area’ ‘now’.” In the conventional method for both searches, the conventional one-dimensional index is used to retrieve the intermediate results that satisfy the one-dimensional condition in the multidimensional condition. Then, the intermediate results are verified individually to determine whether each result satisfies other dimensional conditions. The conventional method cannot always accelerate the search process because it often receives a significant amount of intermediate results. A method has been proposed to use a number of conventional multidimensional indexes to accelerate multidimensional range searches; however, the method has insufficient scalability because it requires significant computation costs to build and maintain indexes.

Therefore, pseudo-candidate key indexing is proposed as a new indexing technology for multidimensional exact match searches. Pseudo-candidate key indexing is a method that indexes AND conditions, and the corresponding results are provided in advance in order to return search results against AND conditions rapidly. However, there are as many AND conditions as the number of attribute combinations, and such number is immense. The important point of the proposed method is to invent a new way for selecting AND conditions to index in such a way that more AND conditions than necessary are not indexed. From the evaluation results, it is confirmed that the proposed method accelerates search queries using AND conditions independently of the amount of M2M data. It is also confirmed that the communication volume and amount of memory for the proposed method is in the logarithmic order of the amount of M2M data; thus, the proposed method has enough scalability. In addition, the load-adaptive algorithm is invented. The algorithm changes the indexed AND conditions adaptively according to updated M2M data.

Furthermore, UBI-Tree is proposed as a new indexing technology for multidimensional

range searches. UBI-Tree is based on R-Tree, which is a conventional indexing method for multidimensional range searches, and it is modified to index various data that have various types and numbers of dimensions. From the evaluation results, it is confirmed that the proposed method can accelerate multidimensional range searches compared to the conventional method. Furthermore, the Load-Adaptive UBI-Tree is proposed. The Load-Adaptive UBI-Tree changes the way in which data is classified adaptively according to the history of search queries. From the evaluation results, it is confirmed that the proposed method improves average latency and throughput by improving the efficiency of frequently used search queries. In addition, the dynamic-help method is proposed as a new load-distribution method for the scalability of UBI-Tree. From the evaluation results, it is confirmed that the proposed method has scalability against an increasing load.

As described above, two new indexing technologies are established in this study as elemental technologies that can help make multidimensional searches efficient for the search function on horizontal-integrated platforms. For future work aimed at practical use, a total system design of horizontal-integrated platforms that include other indexing technologies and functions, full-scale implementation, and confirmation of validity through experimental trials is required.

M2M データのためのインデキシング技術の研究

荒川　豊

概要

近年の通信機器の小型化やネットワークインフラストラクチャの大容量化・低料金化により，M2M（Machine to Machine），すなわち機器間の通信を用いたアプリケーションが普及しつつある．さらに，個々の M2M アプリケーションを独立に実装していたのでは，費用や実装期間が大きくなってしまうため，共通機能，あるいはデバイスやデータそのものを共有化する，水平統合型プラットフォームの必要性が認識されつつある．

本研究では，M2M の更なる普及を促進するため，水平統合型プラットフォームの基本機能であるデータ検索機能の実現を目指すこととした．大規模データに対する効率的な検索には，インデックスが必要不可欠である．また，扱うデータや検索クエリに応じて，適切なインデックスは異なる．そこで， M2M アプリケーションの M2M データ検索クエリに適したインデキシング技術を確立することを，本研究の目的とした．水平統合型プラットフォームの検索機能においては，大量のデータを蓄積しながら，大量の検索処理を実行するため，挿入性能，検索性能ともに高速性とスケール性が求められる．各 M2M アプリケーションは種類によって様々な検索条件により検索を行うが，各検索条件が用いられる頻度を正確に予測することはできず，また頻度は動的に変化すると考えられるため，どのような検索をどれほど効率的に処理可能とすべきかを事前に決めることは困難である．従って，入力データや検索要求の変化にもある程度柔軟に対応可能な負荷追従性が求められると考えられる．

本研究ではまず ，M2M アプリケーションの検索要求について検討・整理した．その結果，5 つの検索パターンが存在することが判明し，特にデバイスの「ID」と測定時刻を示す「時間」の 2 つの次元（属性）を指定する検索条件を用いた 2 次元検索クエリが多く用いられることが分かった．しかしながら，従来のインデキシング技術で対応できない検索クエリという観点からは，本論文中では「その他の多次元検索型」と分類した検索パターン，すなわち多様な M2M データに対する多様な多次元検索クエリが，従来のインデキシング技術では前述した要件，すなわち高速性，スケール性，負荷追従性を実現できない検索要求であることが分かった．M2M データは，「ID」や「時間」，「場所」など，多数の次

元（属性）を含み，多次元検索クエリは，複数の次元について条件指定した検索クエリである．「その他の多次元検索型」をさらに分類すると，多次元完全一致検索と多次元範囲検索に分けることができる．多次元完全一致検索とは，「『2014 年製』の『X 社』の『赤い』『カバン』」を検索条件とする検索クエリのように，モノ・コトに関する複数の属性情報を指定する検索である．多次元範囲検索とは，「『関東地方』で『現在』の積載量が『0.5t 以下』のトラックのリスト」を検索条件とする検索クエリのように，時空間やセンサデータ等，複数次元の値範囲を指定する検索である．多次元完全一致検索，多次元範囲検索いずれの検索においても，1 次元インデックスを用いて中間解を取得し，中間解に対して残りの次元についてフィルタリングを行う従来手法では中間解が膨大になり高速化できない．また多次元範囲検索については，多数の多次元インデックスを用いる従来手法もあるが，インデックスの構築・保持コストが高くスケール性に乏しい．

そこで，多次元完全一致検索に対する新しいインデキシング技術として，準候補キーインデキシング技術を提案した．準候補キーインデキシング技術は，複数の属性情報を指定した検索条件（AND 条件）に対しても高速に検索結果を返却できるよう，AND 条件とそれに該当する検索結果も予めインデックスに入れておく手法である．ただし，AND 条件は組合せの数だけ存在し，その数は爆発してしまうため，必要以上に多くの AND 条件をインデキシングしないで済むよう，インデキシングすべき AND 条件の選択手法を考案した点が本技術の重要な部分である．評価の結果，AND 条件においても M2M データ数に関わらず高速な検索が実現でき，またインデキシングのための通信量や記憶量は M2M データ数に対して log オーダとなり，スケール性があることを確認した．また M2M データの更新に追従して，インデキシングする AND 条件を適応的に変化させていくアルゴリズムを考案した．

また多次元範囲検索に対する新しいインデキシング技術として，UBI-Tree 技術を提案した．UBI-Tree 技術は，従来の多次元範囲検索用インデキシング技術である R-Tree を拡張し，多様なデータ，多様な範囲条件にも対応可能とする手法である．評価の結果，従来技術に比べ，高速な多次元範囲検索が可能であることを確認した．さらに，変化する負荷に対応できるよう，検索クエリの履歴に応じてデータの分類方法を変化させていく Load-Adaptive UBI-Tree 技術を提案した．評価の結果，頻繁に用いられる検索クエリの処理効率を向上させ，平均レスポンスタイムやスループットを向上させることができることを確認した．また，UBI-Tree 技術にスケール性を持たせるため，新しい負荷分散手法として Dynamic-Help Method を提案した．評価の結果，増加していく負荷に対してスケールすることを確認した．

以上のように本研究では，水平統合型プラットフォームの検索機能における多次元検索を効率化する要素技術として，2 つの新しいインデキシング技術を確立することができた．今後，実用化していくためには，他のインデキシング技術，他の機能を含めた水平統

合型プラットフォームとしてのトータルな設計，本格実装，実証実験による有効性確認を行っていく必要があると考えられる．

目次

図目次

第 1 章

序論

1.1 M2M の概要

1.1.1 M2M の歴史

1991 年，米ゼロックス社パロアルト研究所の Mark Weiser 氏が「ユビキタス」の概念を提唱してから，早 20 年以上経過した．英語でユビキタスとは「遍在する」ことを意味し，「ユビキタスコンピューティング」,「ユビキタスネットワーク」,「ユビキタス社会」などの言葉は，我々の身の回りにコンピュータや通信機器などの各種デバイスが遍在し，「いつでも，どこでも，だれでも」がそれらから恩恵を受けることができる環境，社会を指す．

類似の概念を表す言葉として，近年では「M2M」が広く用いられるようになりつつある．M2M とは Machine to Machine の略であり，人間の介在無しに機器同士で行う通信，あるいはそのような通信を用いたシステムを指す．サービス的視点から生じた「ユビキタス」に対し，「M2M」はシステム的な視点から生じた言葉と言えるが，それらが表す世界はよく似たものであると考えられる．

近年の通信機器の小型化やネットワークインフラストラクチャの大容量化・低料金化により，M2M の通信により実現されるアプリケーション（M2M アプリケーション）への期待が高まりつつある．既に M2M を用いた多数のアプリケーションが実現しつつあり，2018 年の M2M 市場は 1 兆円を超えると見積もられている [1]．現在全デバイスの 99.4%は未接続であるが，さらに今後加速度的に接続数は増加し，100 億デバイスが接続されて IoT（Internet of Things），すなわちモノのインターネットが形成され，2020 年には 500 億デバイスが接続され，IoE（Internet of Everything）が形成されると予測されている [2]．

1.1.2 M2M アプリケーション例

近年の代表的な M2M アプリケーションの 1 つとして，北米を中心としたスマートグリ

ッドがある．スマートグリッドは，火力・原子力・風力・太陽光など供給側発電設備と，一般家庭やビルなど需要側機器をネットワークで結ぶ次世代送電網である．従来のような集中制御ではなく，自律分散制御を行うことで需給バランスの最適化を実現し，電力の安定化やコスト最小化を目指すシステムであり，国内でも実証実験が進められつつある．また欧州の交通事故緊急通報システムは，自動車事故発生時に車載の通信ユニット（eCall）が自動で緊急通報を行うシステムである．欧州においては，2015 年 10 月より自動車への搭載が義務化される予定となっている [3]．またブラジルにおいても，M2M を利用した盗難車両追跡システムの導入が進められている．

この他，国内においても建設機械管理 [4]，自動販売機管理 [5]，農業支援 [6]，プローブ情報を用いた交通管理 [7] [8]等が実用化されている．

このように，国内外で多様な M2M アプリケーションが現れつつある．

1.1.3 最近の動向: M2M アプリケーションのための水平統合型プラットフォーム

M2M 市場が急成長しつつあるが，企業が製品・サービスの競争力強化に M2M データを活用する際，システム投資規模が莫大となってしまうことが課題となっている．このため，現状では多くの場合スケールメリットを享受できる一握りの先進的な企業でのみ M2M データが活用されている．従来の M2M アプリケーションは，ひとつの領域に閉じた垂直統合型システムにより提供されてきたため，異なる領域のアプリケーション連携はもとより，既存アプリケーションの拡張においても多くの費用が発生してしまう．また，新規領域において新アプリケーションを立ち上げる場合には，システムを一から構築する必要があり，多くの費用が必要となる．また迅速なサービス開始も難しい．

このため，近年では M2M アプリケーションのための水平統合型プラットフォームの必要性が広く認識されつつある．水平統合型プラットフォームにより，多くの共通的な機能が共有化され，またデバイスやデータを 2 次利用，3 次利用することが可能となるため，低いコストで各種 M2M アプリケーションを実現することができる．また他アプリケーションとの連携も容易となるためアプリケーションの多様化が促進され，さらに迅速なアプリケーション導入が可能となる．欧州をはじめ，米国，アジアなどの地域標準化団体が集まって設立された oneM2M [9] においても，水平統合型プラットフォームに関する標準化が進められつつある．oneM2M では，図 1 に示すようにアプリケーション関連の機能（AE: Application Entity）とネットワーク関連の機能（NSE: Network Serice Entity）との間に M2M の共通サービス機能（CSE: Common Service Entity）が存在し，それらが相互に通

信を行うモデルを基本としている．oneM2M のアーキテクチャが提供するプラットフォーム機能の中心である CSE には，図 1 に示される 12 の基本機能が含まれており，デバイスの発見や管理，アプリケーションの管理，M2M サービス中のデータの管理や認証やセキュリティに関する機能などが提供される [10]．また，既に商用の水平統合型プラットフォームも現れ始めている [6] [11] [12].

また一方，最近では新しい M2M データ収集方法として「参加型センシング」が注目を集めている [13] [14] [15]．参加型センシングとは，各ユーザが自身の保持するスマートフォンに付帯するセンサで取得したセンサデータや，自身で確認した事象の口コミ情報等を持ち寄ることで，低いコストで広い地域をカバーし，多くのセンサデータを収集することを可能とする M2M データ収集手法である．水平統合型プラットフォーム上で共有する大量の M2M データを収集する手段の一つとして期待される．

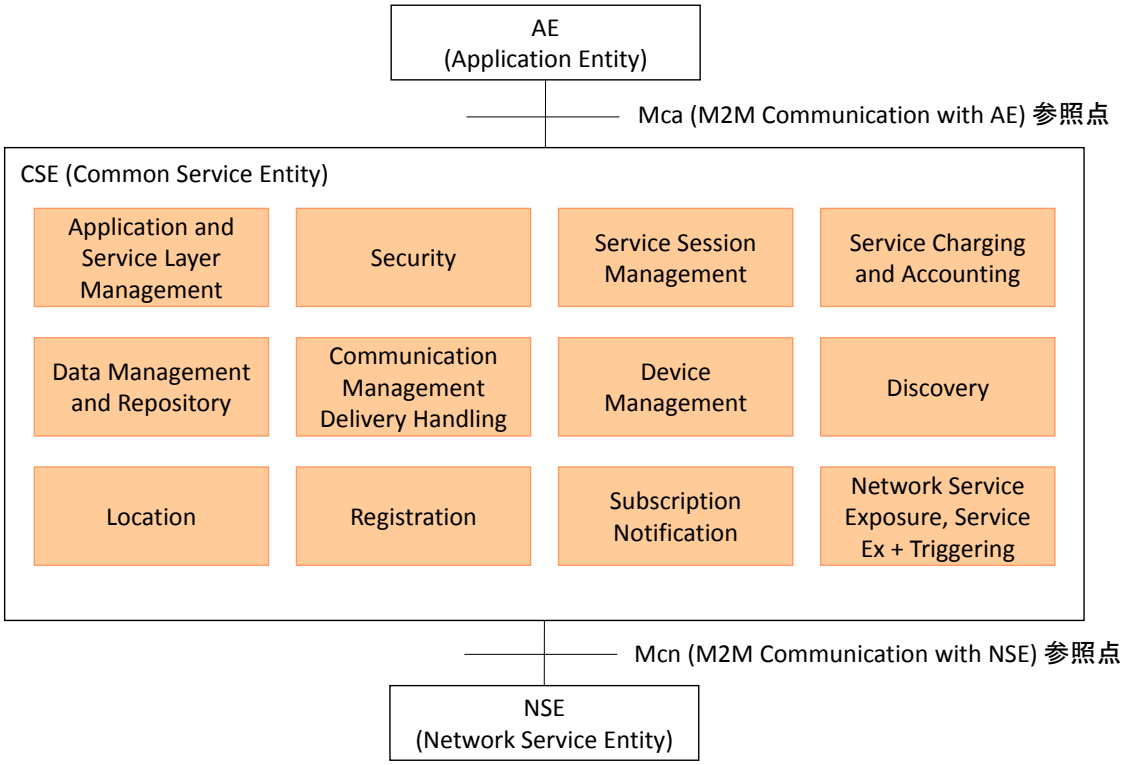


図 1　CSE の基本機能

1.1.4 水平統合型プラットフォームにおける課題

M2M アプリケーションのための水平統合型プラットフォームには，スケール性のあるデータ処理方式，柔軟なアクセス制御，課金，デバイス管理など，多くの機能が求められるが，最も基本的な機能の 1 つに M2M データの検索機能がある．水平統合型プラットフォームの検索機能においては，大量のデータを蓄積しながら，大量の検索処理を実行するため，挿入性能，検索性能ともに高速性とスケール性が求められる．また各 M2M アプリケーションは種類によって様々な検索条件により検索を行うが，各検索条件が用いられる頻度を正確に予測することはできず，また頻度は動的に変化すると考えられるため，どのような検索をどれほど効率的に処理可能とすべきかを事前に決めることは困難である．従って，入力データや検索要求の変化にもある程度柔軟に対応可能な負荷追従性が求められると考えられる．M2M データは，M2M システムを形成する各種デバイス（温度，湿度，スイッチなどの各種センサや，ブザー，モーター，ディスプレイなどの各種アクチュエータなど）と，各種アプリケーションプログラムの間で入出力されるデータであり，多様なセンサデータやログデータ，また操作命令データなどを含む．水平統合型プラットフォームに蓄積した M2M データに対する検索機能は M2M アプリケーションにとって必要不可欠であるが，M2M データは従来のデータと特性が異なるため，上記高速性，スケール性，負荷追従性をもつ検索機能の実現は技術的にチャレンジングである．

M2M データは個々のデータは小さいものの，数が非常に多く，多様性に富むという特徴をもつ．数が多く，多様性に富む従来のデータとしては Web データ等があるが，M2M データはさらに，ユーザやアプリケーションによって必要とされるデータが異なるという特徴がある．あるユーザにとっては，自分の周囲の時間，場所に関する M2M データが有用である場合が多く，当然ながら当該時間，場所はユーザによって異なる．また温度分布を表示するアプリケーションにとっては，温度情報を含む M2M データが有用であり，交通状態を監視するアプリケーションにとっては，各自動車の位置情報等を含む M2M データが有用である．このように M2M データは，使う側によって価値が変わると言える．これに対し，Web データはユーザ毎に多少の違いはあるものの，あるユーザにとって価値のある Web データは多くのユーザにとっても有用である．このため Web 検索エンジンは，キーワードを指定された検索クエリに対し，当該キーワードを含む，多くのユーザにとって有用な Web データを回答するよう設計されている．「多くの人にとって有用なデータ」という暗黙の検索条件が前提とされていると見做すこともできる．逆に言えば，Web 検索エンジンにおいて，重要度の低い「一部の人だけに有用なデータ」を探すことは困難である．たとえ求めるデータに含まれるキーワードを複数指定したとしても，当該データの重要度が低ければ当該データを検索結果に見出すことは難しい．水平統合型プラットフォーム上の M2M データを活用する上では，従来の Web 検索エンジンとは異なるアーキテクチャで検索機能を実

現する必要がある．

大量データの検索において，インデックス（索引）は必要不可欠であり，技術的な肝である．例えば Web 検索エンジンにおいて，どのようなインデキシング技術を用いるか（どのような構造のインデックスを用い，どのようにデータを索引付けするか）は Web 検索エンジンの機能や性能を決定する重要な要素である．M2M データは，上述したようにユーザやアプリケーションによって必要とされるデータが異なるという特徴があるため，Web 検索エンジンで用いるインデキシング技術をそのまま適用することはできない．M2M データは数値データを多分に含み，範囲検索が必要となる場合も多いため，その意味でもキーワード検索を得意とする Web 検索エンジンは不適である．M2M データに対する検索機能においても，M2M データの特徴に応じたインデキシング技術が求められる．

1.2 インデキシング技術の概要

本節では，インデキシング技術について概説する．まず，インデキシング技術とは何か，またその具体例について述べ，次にインデキシング技術の適用方法の 1 例として，最も代表的な適用先の 1 つであるリレーショナルデータベース（RDB; Relational Database）の機能概要とその中でのインデックスの使われ方について紹介する．

1.2.1 インデキシング技術とその具体例

1.2.1.1 インデキシング技術概要

インデックス（索引）とは，複数のデータの中から，目的のデータを高速に見つけだすためのデータ構造である．検索を高速化するという基本的な役割を持つことから，インデックスの適用範囲は広い．データベースにおける検索の高速化に用いられる他にも，各種プログラミング言語で実装されている連想配列などのコンテナライブラリや，ファイルシステムなど，多くのシーンで活用されている．

データの種類，あるいは「目的のデータ」の指定方法，すなわち検索条件の種類によって，数多くの種類のインデックスが存在する．逆に言えば，インデックスの利用者は，インデックスの適用先により使用するインデックスの種類を適切に選択する必要がある．データベース設計者・運用者にとって，データや検索要求に応じたインデックスの選択は重要な仕事の 1 つである．

本節では，数あるインデックスのうち，最もよく使われている代表的なインデックスとして，ハッシュテーブル（ハッシュインデックス）と B-Tree について説明する．

1.2.1.2 ハッシュテーブル

ハッシュテーブル（Hash Table）は，ハッシュ関数を用いることでデータの高速な検索を可能とするインデキシング技術である．データの個数よりも大きめのメモリ領域を活用することで，データの挿入・削除・検索をデータ数に依存しない一定時間，$O(1)$の処理コストで実行することができる．

ハッシュテーブルの構造を図 2 に示す．図 2 は，従業員番号をキーとし，従業員名を検索可能としたハッシュテーブルの例である．ハッシュ関数には，従業員番号を 7 で割った余りを返す関数を用いている．例えば従業員番号が 1320 の従業員名を検索する場合，図に示したようにまず 1320 をハッシュ関数にかけ，その戻り値（ハッシュ値と呼ぶ）4 を得る．次に，ハッシュテーブルを参照し，該当する行を参照すれば，従業員番号 1320 の従業員名「吉田」を得ることができる．またデータを登録・更新する際にも，同様の手順により実行することが可能である．

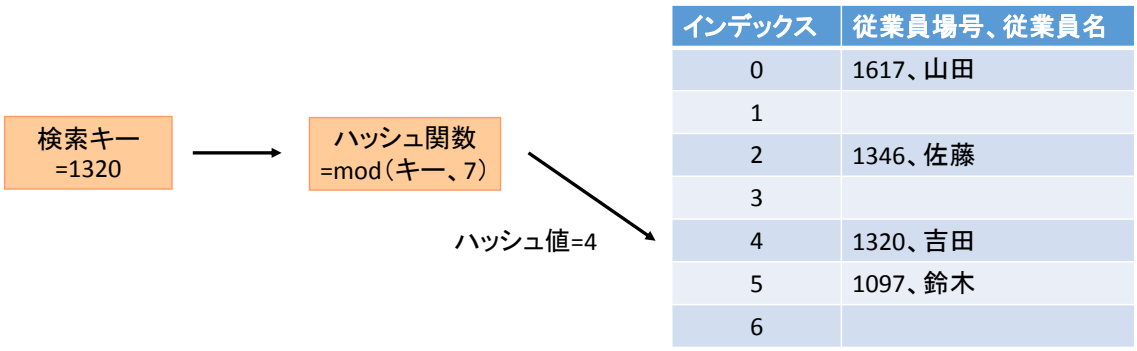


図 2　ハッシュテーブルの構造

ハッシュテーブルはその原理から，異なるデータでもハッシュ値が衝突する場合がある．衝突への対処方法としては，同じハッシュ値のデータをリストで管理する手法や，次のハッシュ値用の領域を利用する手法などがあるが，いずれにしても衝突が多発する場合には検索効率が低下してしまう．そこで，図 2 には 7 行しかないハッシュテーブルを示したが，実際にはデータ数よりも大きなハッシュテーブルを用いることで，データの衝突を抑える．データ数の増加に応じて，ハッシュテーブルサイズを拡大する手法もある．また衝突がなるべく発生しないよう，大量のデータのキーに対して，ハッシュテーブル（ハッシュ空間）全体に均一に割り振られるようなハッシュ関数が用いられる．

また，ハッシュテーブルは完全一致検索を高速化する手法であり，範囲検索に用いることはできない．

1.2.1.3 B-Tree

B-Tree は，データを木構造に整理・格納しておくことでデータの高速な検索を可能とするインデキシング技術である．B-Tree は多分木，すなわち木構造を構成する各ノードから複数の枝が伸びる木である．各ノードが最大 m 本の枝を持つ場合，これをオーダ m の B-Tree と呼ぶ．また B-Tree は平衡木（またはバランス木），すなわちどのリーフノードも，同じ高さになるようにデータが格納された木である．これにより，格納したどのデータを検索する場合にも，同じ処理コストで実行可能となる．n 個のデータを格納した場合，1 つのデータを検索するのに要する処理コストは，どのデータについても $O(\log_m n)$である．

B-Tree の構造を図 3 に示す．図 3 は，ハッシュテーブルの例と同様に，従業員番号をキーとし，従業員名を検索可能とした B-Tree の例である．オーダ 3 の B-Tree であり，各ノードは最大 3 つのポインタと，最大 2 つのキー値をもつ．各ポインタは，子ノードあるいはデータを指している．あるキー値に対し，左側のポインタで辿られる部分木には，当該キー値よりも小さなキー値をもつデータが格納されており，右側のポインタで辿られる部分木には，当該キー値と同じか，より大きなキー値をもつデータが格納されている．データを検索する際には，ルートノードからポインタで示される枝を辿っていく．例えば従業員番号が 1320 の従業員名を検索する場合，まずルートノードを参照し，キー値 1230 が格納されていることを確認する．検索するキー値 1320 は 1230 よりも大きいことから，キー値 1230 の右側のポインタで示される枝を辿っていく．次のノードを参照すると，1320 は 1246 より大きく，1346 より小さいことから，当該ノードの中央のポインタで示される枝を辿る．リーフノードでも同様にポインタを辿れば，従業員番号 1320 の従業員名「吉田」を得ることができる．

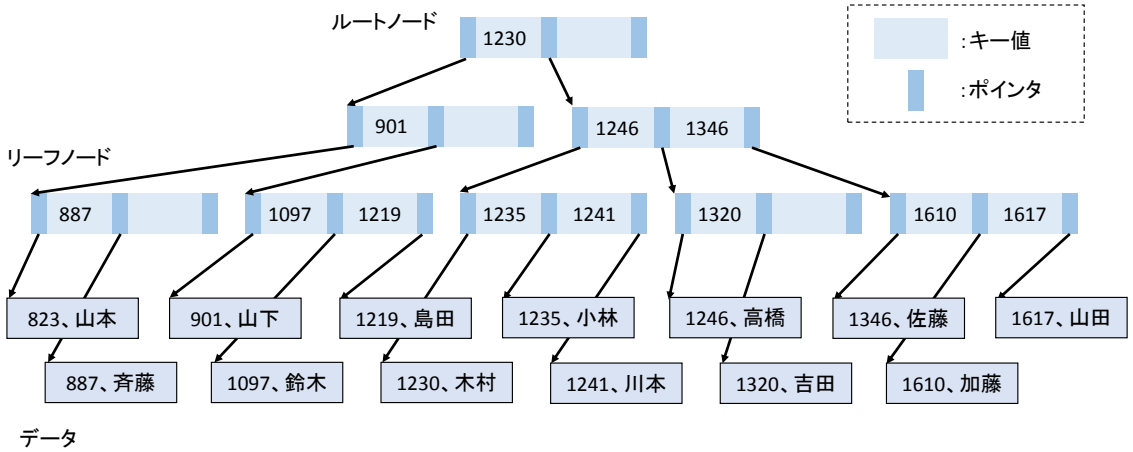


図 3　B-Tree の構造

データの登録においても同様に，ルートノードからデータを登録すべきリーフノードへ枝を辿っていく．ポインタ数がノードの許容量（オーダ）を超えた場合には，ノードの分割処理を行う．またこれにより，木のバランスが悪くなった場合（リーフノードによって高さが異なるようになった場合）には，バランスをとるよう調整する処理を行う．

B-Tree の検索処理コストはハッシュテーブルよりも大きくはあるが，有用なデータ構造として広く利用されている．また各データはキー値でソートされた形で格納されるため，効率的な範囲検索が可能である．

1.2.2 インデキシング技術の適用例

データベースの運用・管理のためのシステム，およびそのソフトウェアをデータベースマネジメントシステム（DBMS; Database Management System）と呼ぶ．最も広く利用されているデータベースであるリレーショナルデータベース（RDB）の DBMS，すなわち RDBMS は，インデキシング技術の代表的な適用先と言える．RDBMS の一般的な機能要素を図 4 に示す [16].

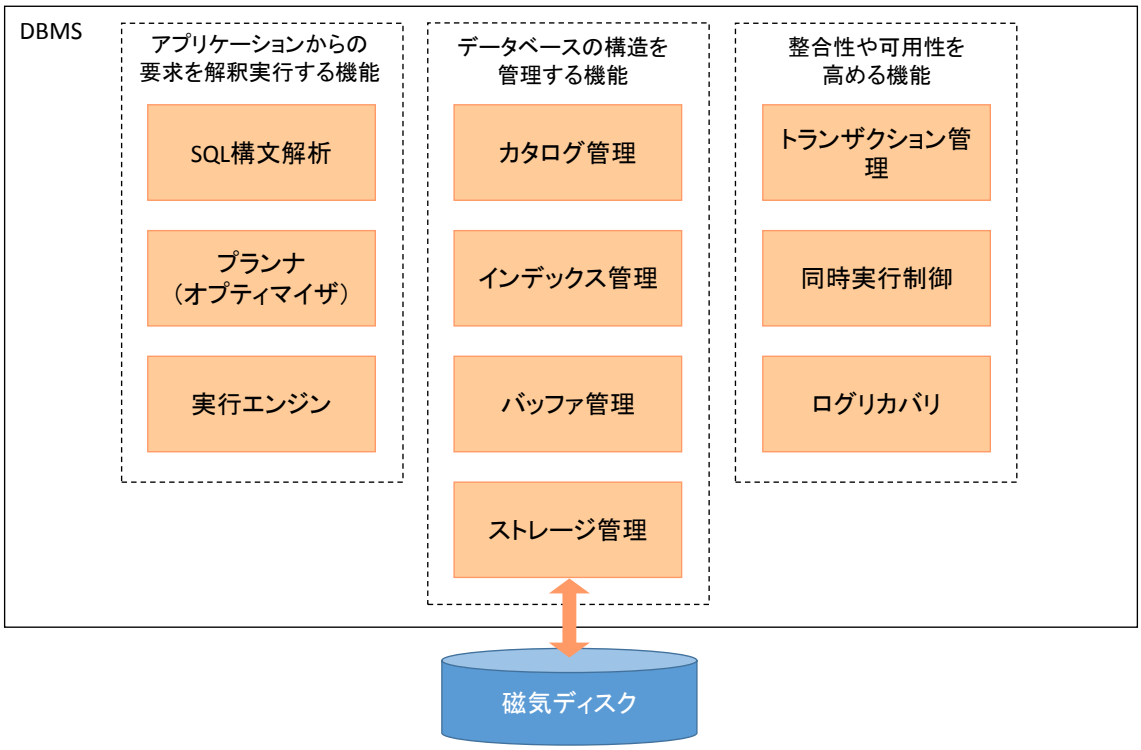


図 4　RDBMS の機能要素

図 4 に示した RDBMS を構成する各機能の概要はそれぞれ以下の通りである．

- SQL の構文解析

  アプリケーションから文字列として入力された SQL をパースし，内部形式に変換する．

- プランナ（オプティマイザ）

  構文解析した SQL の内部形式をもとに，実行プランを作成する．予め決められたルールに従い実行プランを作成する方式や，データの統計情報などに従い実行プランを作成する方式が存在する．

- 実行エンジン

  プランナが作成した実行プランに従い，処理を実行する．

- カタログ管理

  データベース内のテーブルやビュー，それらのカラムや属性，制約条件，アクセス権限など，データベースの論理構造を管理する．

- インデックス管理

  テーブルから目的の値を含む行を検索する際，検索を高速化するため予め構築したインデックスを参照したり，データ変更に伴いインデックスを更新したりする．

- バッファ管理

  テーブルやインデックスなど，データベースに格納されているデータを読み書きするために，必要な部分をディスクから読んだり，書き戻したりする．並列的に実行される各処理がバッファを共有することで，実際のファイル入出力の回数を減らすことができる．

- ストレージ管理

  データベースを構成するファイルの構造を管理する．ファイルとの入出力の単

位となるブロックサイズや，どのファイルにどのテーブルやインデックス，カタログを格納しているかの管理，ファイル入出力のシステムコール読み出しなどを行う．

- トランザクション管理

  アプリケーションにとって意味のある処理単位で，データベースを更新できるようにする機能である．データベースやアプリケーションの故障などが発生した場合でも，データベースを整合性のある形で維持することを可能とする．

- 同時実行制御

  複数のアプリケーションやユーザから同時に SQL を受け取った場合，同時実行しつつ，互いの処理が干渉しないよう制御を行う．

- ログリカバリ

  データベースに異常が発生した場合に，データを復元するための情報となる，バックアップとログを記録する．

RDBMS によって，上記機能群の実装には様々なバリエーションが存在し，また上記機能群の全てが実装されているとは限らない．またリレーショナルではない DBMS においても同様に，上記機能群の多くが必要とされるが，やはりその全てを必要とするとは限らない．例えば，近年多くの実装が現れている Key-Value ストアにおいては，単純なデータの読み書き（Put，Get）程度の API のみをサポートし，複雑な SQL の構文解析は不要である場合が多い．またパーソナルユースのデータベースにおいては，同時実行制御機能は不要であるため実装されていない場合もある．また性能を重視し，トランザクション管理を敢えて完全には行わない DBMS もある．

インデックス管理機能も，DBMS にとって必須とは言えないが，データ検索は DBMS の基本機能であり，その高速化のためのインデックス管理機能は通常備えられている．ただし，インデックスには様々な種類のものが存在し，サポートするインデックスの種類は DBMS によって異なる．

水平統合型プラットフォームにおけるデータ管理機能は DBMS に相当し，本論文で提案する新しいインデキシング技術は，当該インデックス管理機能においてサポートされる新しい種類のインデックスとして位置づけられる．なお，図 4 からも分かるように，インデックス管理機能はストレージ管理機能とは別であり，ファイル上に蓄積されたデータ群に

対し，複数の検索条件に対して高速な検索ができるよう複数のインデックスを構築したり，検索条件の変化に伴い別の種類のインデックスに変更したりすることも可能である．しかしながら，インデックス管理機能とストレージ管理機能とは性能を向上させる上で密接な関係をもつため，用いるインデックスによって，ストレージ管理機能が司るファイル構造が規定される場合もある．

1.3 本論文の概要

1.3.1 本論文の目的

1.1 で論じた M2M の概要を鑑み，M2M データの活用を促進する水平統合型プラットフォームにおける検索機能のための新しいインデキシング技術の確立を目指す．まず，M2M アプリケーションの検索要求にはどのようなものがあるかを検討・整理する．さらに，従来のインデキシング技術では効率化できない検索要求に対し，新しいインデキシング技術を提案し，実験・考察による評価を行う．

1.3.2 本論文の成果と構成

本論文では，M2M アプリケーションの検索要求について検討・整理し，従来のインデキシング技術では効率化できない検索要求である多次元検索が必要とされることを明らかにした．また，当該多次元検索には多次元完全一致検索と多次元範囲検索が含まれるが，多次元完全一致検索のための新しいインデキシング技術として「準候補キーインデキシング」技術を，また多次元範囲検索のための新しいインデキシング技術として「UBI-Tree」技術を，それぞれ確立した．本論文の構成は以下の通りである．

第 2 章では，多様な M2M アプリケーションの検索要求について整理し，M2M データに対する多次元完全一致検索と多次元範囲検索が必要とされていると結論付けられることを述べる．まず M2M アプリケーションの検索要求について検討・整理した結果，M2M アプリケーションの検索要求には大きく 5 つの検索パターンが存在することが判明した．さらに，検索パターンの 1 つである多様な M2M データに対する多次元検索は，これまでにない検索要求であり，従来のインデキシング技術では必ずしも効率化できないことが判明した．当該多次元検索は，例えばセンサ値そのもの，あるいはその他属性情報等を検索条件に指定した多次元検索であり，多次元完全一致検索と多次元範囲検索が含まれる．従来の 1 次元インデキシング技術であるハッシュテーブルや B-Tree を用いて多次元検索を

効率化する場合，多次元検索条件（AND 条件）に含まれる 1 次元の条件で検索した検索結果（中間解）に対し，他の次元の条件を満たすかどうかを検査する手法（フィルタリング），または，各次元の条件で検索し，それぞれの中間解を照らし合わせ，全ての中間解に出てくる検索結果を抽出する手法（マージ），あるいはそれらを組み合わせた手法が用いられる．しかしながらいずれの手法においても，中間解の量が大規模となる場合には，フィルタリングやマージに要する処理コストが大きくなってしまうという問題がある．

第 3 章では，多次元完全一致検索に対する新しいインデキシング技術として提案する，準候補キーインデキシング技術について述べる．準候補キーインデキシングは，従来の 1 次元インデキシング技術であるハッシュテーブルをベースとするが，AND 条件を用いた場合にも高速に検索結果を返却できるよう，AND 条件とそれに該当する検索結果も予めインデックスに入れておく手法である．図 5 に，多次元完全一致検索（キーワード検索）技術における準候補キーインデキシング技術の位置付けを示す．図 5 に示すように，扱うデータ量が小さい多次元完全一致検索は，デスクトップ検索や LAN 上のドキュメント検索において用いられ，全文検索技術等が活用され既に多くの製品が存在している．また扱うデータ量が大きく，かつ各データの重要度の偏りが大きい場合の多次元完全一致検索は Web 検索において用いられる．1.1.4 節で述べたように，Web データはあるユーザにとって価値のある Web データは多くのユーザにとっても有用である．すなわち，各データの重要度の偏りが大きく，このため使用される検索条件の偏りも大きい．このような場合，例えば検索結果をキャッシュすることで検索を効率化する手法 [17]が有効となる．一方，M2M データは使う側によって価値が変わり，各データの重要度の偏りが小さい．準候補キーインデキシングは，これまでのインデキシング技術とは異なり，検索結果がある一定数（T 個）以下となる検索条件であればどのような検索条件でも効率化できることから，M2M 検索における多次元完全一致検索に適していると言える．

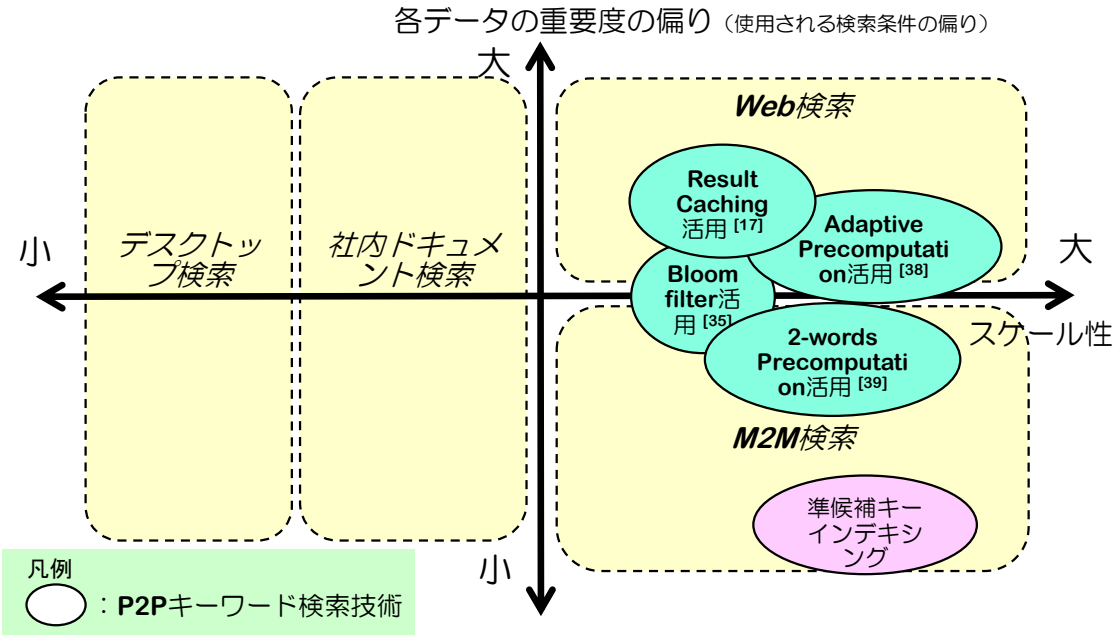


図 5　多次元完全一致検索技術における準候補キーインデキシング技術の位置づけ

また第 4 章では，多次元範囲検索に対する新しいインデキシング技術として提案する，UBI-Tree 技術について述べる．UBI-Tree は，多次元範囲検索のための従来技術である R-Tree をベースとするが，多様なデータ形式（スキーマ）のデータを含む M2M データ（スキーマレスデータ）と，M2M データに対する多様な範囲検索条件に対応できるよう拡張した手法である．図 6 に，多次元範囲検索技術における UBI-Tree 技術の位置付けを示す．図 6 に示すように，現在最も利用されているリレーショナルデータベース（Relational Database; RDB）は，スキーマが統一された帳票データ等を主な対象としている．RDB においては，範囲検索の効率化には多くの場合 B-Tree（あるいはその派生技術）が用いられ，また多次元範囲検索には R-Tree（あるいはその派生技術）等が用いられる場合もある．一方，XML-DB の他，近年スキーマレスな（不定形な）データを扱うデータベースとして CouchDB や MongoDB が注目を集めている．これらのデータベースでは，1 次元範囲検索の効率化に B-Tree が用いられている．M2M データはスキーマレスなデータであり，また代表的な M2M データであるセンサデータ等は多次元連続値をもつため，スキーマレスデータに対する多次元範囲検索が必要となる．UBI-Tree は，これまでのインデキシング技術では対応できなかった，当該検索の効率化を実現する技術である．なお，多次元範囲検索のための従来のインデキシング技術は，スキーマフルなデータに対するものであり，画像・映像の特徴量による検索を可能とするマルチメディアデータベース等において用いられる．

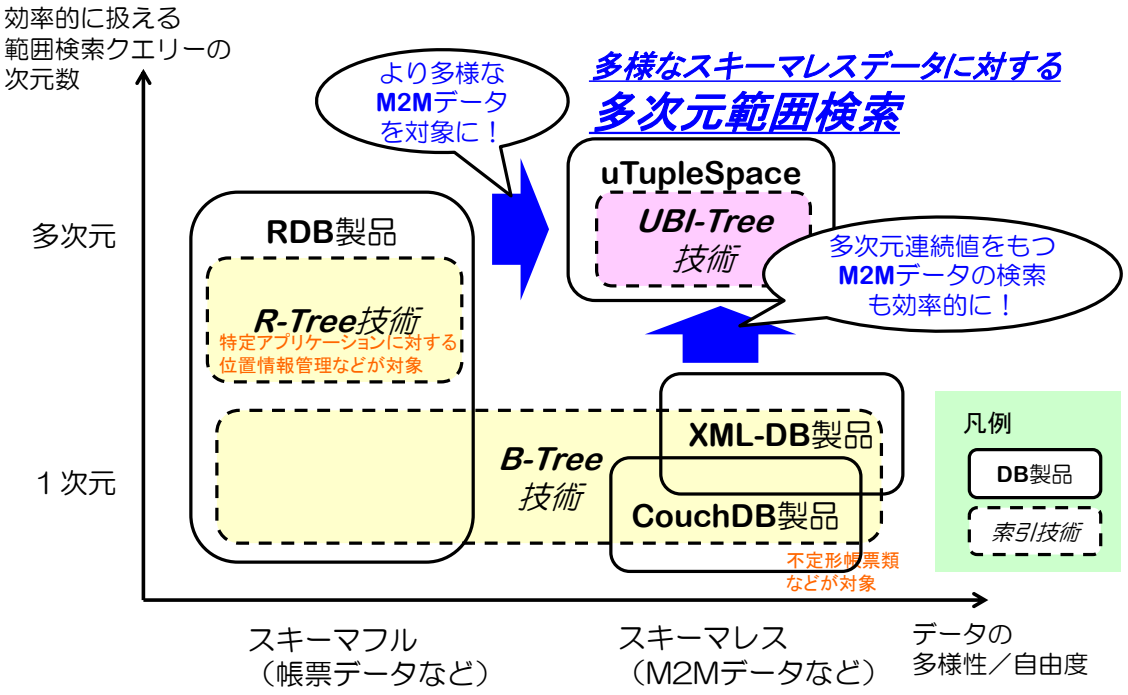


図 6　多次元範囲検索技術における UBI-Tree 技術の位置づけ

最後に第 5 章で本論文のまとめと今後の課題について述べる.

第 2 章

M2M アプリケーションの検索要求

本章では，想定される各種 M2M アプリケーションに基づき，水平統合型プラットフォームにおいてどのような検索が求められるかを議論する．

まず，電気通信大学 市川研究室において検討が進められてきた M2M アプリケーションである蓄電池管理アプリケーションを例題として，検索機能への要求条件を検討し，次にその他各種 M2M アプリケーションについて，広く検討・整理を行う．

2.1 蓄電池管理アプリケーション

電気通信大学 市川研究室ではこれまで，物流システムを用いて多数の蓄電池を流通させることにより，電力網を敷設することなく BoP(Bottom Of Pyramid; 開発途上地域に多く存在する低所得層）地域において電力インフラを実現する手法を検討してきた．図 7 にシステム構成概要を示す．このような ICT 支援は，人道的・社会的意義はもちろんのこと，新たなビジネス市場開拓という意味でも重要である．

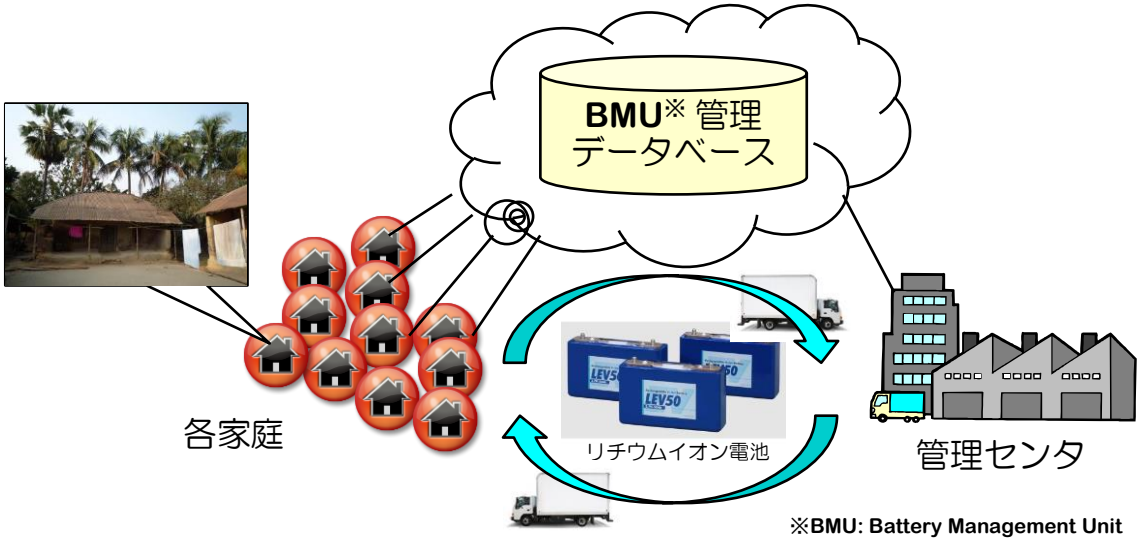


図 7　蓄電池管理システム概要

図 7 に示す蓄電池管理システムは，特定の地域に集中整備したインフラ基盤上にデータセンターを配備し，そこで蓄電池管理アプリケーションを動作させる．常時通信網が整っていないため，携帯回線や手持ち運搬により多様な M2M データを集約し，蓄電池（BMU; Battery Management Unit）の充電率予測など各種用途に活用する．

蓄電池管理アプリケーションによる M2M データを用いた蓄電池管理では，以下の管理が行われる．

- 利用
  - SOC（State of Charge; 充電率）推定
  - 充放電制御：長寿命化，安全確保
- 再利用
  - 蓄電池の評価
- 安全確保
  - 事故防止
  - 事故原因分析

各種管理において，蓄積された M2M データが検索・利用される．詳細については Li イオン電池管理に関する資料（付録 A）を参照されたい．このうち，最も高い頻度で実施され

るのはSOC推定であり，日々，個々の蓄電池について充放電流や電圧，温度などの履歴情報を元に推定を実施する必要がある．このときのM2Mデータ検索条件には，蓄電池IDと，時間の範囲が指定される．このことから，＜ID，時間＞の2次元を検索条件に用いた2次元範囲検索が効率的に実施できることを要求条件として挙げることができる．また例えば事故防止のために異常となった蓄電池を発見する検索は，SOC推定に比べ低い頻度で実施され，このときは＜時間，緯度，経度，温度＞の4次元範囲検索が用いられる．このように各種管理が実施される頻度は異なり，また利用する検索条件も別々のものとなる．

以上より，蓄電池管理アプリケーションにおけるM2Mデータ活用方法を考慮すると，実施頻度の異なる種々の多次元範囲検索が可能であることが検索機能への要求条件として挙げられることが分かる．また特に，＜ID，時間＞を検索条件とした2次元範囲条件を効率的に実施できる必要があると考えられる．

2.2 その他M2Mアプリケーション

蓄電池管理アプリケーションにおける考察に倣い，運送・物流，ビル，ユーティリティ，農業など各業界において想定されるM2Mデータを活用するM2Mアプリケーションそれぞれについても，必要とするM2Mデータと，その検索のために用いる検索条件を検討した．表1に結果をまとめる．表1において，必要とするM2Mデータは二重枠で，その検索のために用いる検索条件は色付き網掛けで示されている．例えば表1の冒頭に記載した車両の運行管理アプリケーションであれば，「地理的条件・値」が二重枠，「端末ID」と「時間的条件・値」が色付き網掛けで示されている．これは特定車両の現在の位置，すなわち最も現在に近い位置データを調べるために，当該車両に搭載した端末のIDと時間（現在時刻）を検索条件とした検索を実施することを示している．

表 1　各種 M2M アプリケーションが必要とする M2M データと，その取得のために用いる検索条件

業界	M2Mアプリケーション	主な端末搭載対象物	M2Mアプリケーションが求める値	M2Mアプリケーションが用いる検索条件と求める値（色付き網掛けが検索条件、二重枠が値）						検索パターン
				端末ID（address）	時間的条件・値	地理的条件・値	その他の静的条件・値	その他の動的条件・値	監視型（イベントドリブン）	
運送・物流	運行管理（問合せ型）	自動車（トラック）、オートバイ、列車	特定（あるID）の車両の現在地		現在					2
			特定の車両のある時刻での位置		一時点					2
			特定の車両の移動履歴		範囲					2
			特定の車両の配送先、配送予定時刻							1
			遅れが出そうな車両のIDリスト		現在					4
			idol状態の車両IDリスト		現在					4
	車両台帳管理	自動車（トラック）、オートバイ、列車	なんらかの条件を満たす車両のIDリスト 「〇社××号車（IDとは異なる別名）」「〇〇を積んだトラック」「〇〇線の列車」							4
	危険物配送トラックの事故速報	危険物配送トラック	事故を起こしたトラックのID,位置、積載物、状態（映像）							5
ビル	セキュリティと監視システム	ビル・家屋の入り口や敷地（センサ）	特定の場所の現在のセンサ値（映像含む）		現在					2
			特定の場所の特定の時刻のセンサ値		一時点					2
			ある値となったセンサ値と場所と時刻							5
			ある閾値を越えたセンサ値と場所と時刻							5
			特定のセンサ値の組合せorパターンがあったセンサ値と場所と時刻							5
			ある範囲にあるセンサの現在のセンサ値 「5F西側の映像」「全エレベータ内の映像」		現在					3
	電気・ガスなどの集中管理	各種メーター（ビル）	特定のメーターの現在の値		現在					2
			特定のメーターのある時刻での値		一時点					2
			特定のメーターの変化履歴		範囲					2
			ある閾値を越えたセンサ値と場所と時刻							5
	設備の故障監視・状況の通知	ボイラー、貯水タンク、電気設備	故障を起こした設備のID、位置							5
			特定の設備の状態（温度、貯水量、電圧など）		現在					2
			特定の状態の設備のIDリスト 「〇度以上のボイラー」「〇〇ℓ以下の貯水タンク」							4
	遠隔制御	空調、照明	特定の設備の位置、種別、操作方法							1
	設備台帳管理		なんらかの条件を満たす設備のIDリスト 「特定のビル内で点灯している照明リスト」							4
ユーティリティ	高圧受変電設備の漏電監視	変電設備	故障を起こした設備のID、位置							5
			特定の設備の状態（温度、電圧など）		現在					2
			特定の状態の設備のIDリスト 「〇度以上になっている設備」「〇社〇年製の変圧器」							4

	給排水設備故障監視、貯水池濃度監視	給排水設備、貯水池	故障を起こした設備のID、異常値を示した貯水池のID							5
			特定の設備・貯水池の状態（温度、貯水量など）		現在					2
	風力発電機の監視	風力発電機	故障を起こした発電機のID、位置							5
			特定の発電機の状態（温度、電圧など）		現在					2
			特定の状態の発電機のIDリスト 「発電量が〇W以下になっている装置」							4
	車などの電柱への衝突検知	電柱	衝突された電柱のID、位置、映像、状態							5
農業	生育環境監視（温度、日照時間、雨量）	温度計等のセンサ（ビニールハウス）	特定のセンサの現在の値		現在					2
			特定のセンサのある時刻での値		一時点					2
			特定のセンサのある期間での値リスト		範囲					2
			ビニールハウス・畑内単位でのセンサの現在の値リスト		現在					3
			ビニールハウス・畑内単位でのセンサのある時刻での値リスト		一時点					3
			ある期間における異常値になったセンサ値と時刻と位置		範囲					4
			ある期間におけるビニールハウス・畑単位での統計値（平均、最高値とその時間、最低値とその時間）		範囲					3
	盗難防止	畑（センサ）	特定のセンサの現在の値（映像）		現在					2
			特定のセンサのある時刻での値		一時点					2
			特定の農作物が外部へ持ち出された際の時刻とその映像							5
			ある期間において畑へ侵入者があった時刻とその映像		範囲					5
	盗難防止	家畜（一把握）	特定のセンサの現在の値（映像、位置）		現在					2
			特定のセンサのある時刻での値		一時点					2
			特定の家畜が外部へ持ち出された際の時刻とその映像							5
			ある期間において牧舎・牧場へ侵入者があった時刻とその映像		範囲					5
	昆虫防除（捕らえた虫をカウント）	防除器	特定の防除器の現在のセンサ値（捕らえた虫の数）		現在					2
			特定の防除器のある時刻のセンサ値		一時点					2
病院・医療	妊婦などの位置通知（タクシー等のかけつけサービス）	妊婦、障害者	異常値を示した（陣痛が始まった、倒れた）人の位置							5
	処方箋薬局の待ち時間監視	薬局窓口	特定窓口に対する顧客数の現在値		現在					2
			特定窓口に対する顧客数のある時刻の値		一時点					2
	在宅ヘルスケア（家庭にあるものから健康に関する情報を送信）	体重計、血圧計	自分の状態に関する最近測った値と日時		現在					2
			自分の状態に関する特定の時間における値		一時点					2
			自分の状態に関するある期間における値リスト		範囲					2
			自分の状態に関するある期間における平均値		範囲					2
			全被測定者の平均値							4
			ある条件（年齢・性別）を満たす被測定者の平均値							4
		体温計	自分の状態に関する最近測った値と日時		現在					2
			自分の状態に関する特定の時間における値		一時点					2
			自分の状態に関するある期間における値リスト		範囲					2
			自分の状態に関するある期間における平均値		範囲					2
			全被測定者の平均値							4
			ある条件（年齢・性別）を満たす被測定者の平均値							4

行政と自治体	安否確認	アウトドア（登山、スキー等）	特定の人の現在の位置、状態（体温、体勢など）		現在					2
	運行管理（問合せ型）	パトロールカー、ロードサービスなどのサービスカー	特定の車両の現在地		現在					2
			特定の車両のある時刻での位置		一時点					2
			特定の車両の移動履歴		範囲					2
			なんらかの条件を満たす車両のIDリスト 「現在ある地域にいる」「特定の機材を積んでいる」							4
	防犯対策（誘拐、迷子）	子供（位置情報）	特定の子供の現在地、映像		現在					2
	ダム・河川管理	ダム・河川・ため池の雨量、水位情報等	特定の場所での雨量、水位の現在値		現在					2
			特定の場所での雨量、水位のある時刻の値		一時点					2
			異常値となった場所、値							5
			特定の場所での雨量、水位の特定の期間の指定した時間間隔での値リスト		範囲					2
	水質管理	河川、湖沼、浄水場、給水栓	特定のセンサでの水質の現在値		現在					2
			特定の範囲での水質の現在値（平均値、最高値、最低値）		現在					3
			異常値となったセンサの場所、値							5
			特定のセンサでの水質の特定の期間の指定した時間間隔での値リスト		範囲					2
	上下水道設備管理	マンホールポンプ、中継ポンプ場、上下水道	特定箇所（設備）の現在の状態値（作動状況、水流）		現在					2
			特定箇所（設備）のある時刻の状態値（作動状況、水流）		一時点					2
			異常値となったセンサの場所、値							5
	土砂足外、地すべりなどの地質管理	山の斜面（センサ）	特定箇所の現在値（歪み？傾斜？水分量？）		現在					2
			特定箇所の特定の時間の値		一時点					2
			特定箇所の特定の期間の指定した時間間隔での値リスト		範囲					2
	気象観測	観測装置（気圧、気温、風向、風速、降水量、日照時間、日射量）	特定のセンサの現在値		現在					2
			特定のセンサのある時刻での値		一時点					2
			特定の地理的範囲にあるセンサの現在値（平均値、最高値、最低値）		現在					3
			特定の地理的範囲にあるセンサの特定の期間の指定した時間間隔での平均値、最高値、最低値リスト		範囲					3
	火山噴火監視	火山（センサ）	特定箇所の現在値（特定気体の濃度、温度）		現在					2
			特定箇所の特定の時間の値		一時点					2
			特定箇所の特定の期間の指定した時間間隔での値リスト		範囲					2
			異常値となったセンサの場所、値							5
	花粉観測	花粉観測装置（レーザー光線によるかうんと）	特定箇所の現在値		現在					2
			特定箇所の特定の時間の値		一時点					2
			特定箇所の特定の期間の指定した時間間隔での値リスト		範囲					2
			異常値となったセンサの場所、値							5
	仮出所者の遠隔監視（逃亡防止）	仮出所者（位置情報）	特定の仮出所者の現在地、映像		現在					2
			特定の仮出所者の特定の時刻での位置、映像		一時点					2

交通	運行管理、時刻表（時間通りか否か）	公共交通機関の車両	特定の車両の現在地（路線名、駅名とその駅からの距離、線路IDなど）		現在					2
			特定の車両のある時刻での位置		一時点					2
			特定の車両の移動履歴		範囲					2
			特定の車両の運行予定情報							1
			遅れが出そうな車両のIDリスト							4
			車両間隔が接近した2車両の距離と位置、車両ID							4
			なんらかの条件を満たす車両のIDリスト							4
	運行管理（勤怠管理、燃料使用量、CO2ガスなどの排出量把握）	タコメーター（タクシー）	特定のタクシーの現在の値		現在					2
			特定のタクシーのある時刻での値		一時点					2
			特定のタクシーの特定期間の一日あたりの値リスト		範囲					2
			予定と大幅に異なる値だったタクシーとその値のリスト							4
			なんらかの条件を満たす車両のIDリスト条件を満たす車両のIDリスト 「運転時間が〇時間以下」「燃料使用量と売上金額に不整合がある」など							4
	駐車場・洗車場での車両の出入り情報のリアルタイムでの管理（空車情報）	駐車場・洗車場	特定の駐車場・洗車場（1枠）の現在の使用状況		現在					2
			特定の駐車場・洗車場（1枠）の特定の時刻での使用状況		一時点					2
			特定の駐車場・洗車場（1枠）の時間帯別使用率		範囲					2
			特定の駐車場・洗車場（1枠）の日別使用率		範囲					2
			特定の駐車場・洗車場（1枠）の曜日別使用率		範囲					2
			特定の駐車場・洗車場（1枠）の季節別使用率		範囲					2
			ある駐車場・洗車場全体における現在の使用状況（〇枠使用中、△枠空き）		現在					3
			ある駐車場・洗車場全体における特定の時刻での使用状況		一時点					3
			ある駐車場・洗車場全体の時間帯別使用率		範囲					3
			ある駐車場・洗車場全体の日別使用率		範囲					3
			ある駐車場・洗車場全体の曜日別使用率		範囲					3
			ある駐車場・洗車場全体の季節別使用率		範囲					3
			ある駐車場・洗車場全体における現在空いている枠のIDリスト		現在					3
	駐車場・洗車場でのつり銭、売上情報	料金精算機	特定の駐車場・洗車場（1枠）の1日の売上		現在					2
			特定の駐車場・洗車場（1枠）の曜日別売上		一時点					2
			特定の駐車場・洗車場（1枠）の月別売上		範囲					2
			特定の駐車場・洗車場（1枠）の年別売上		範囲					2
			駐車場・洗車場全体の1日の売上		現在					3
			駐車場・洗車場全体の曜日別売上		一時点					3
			駐車場・洗車場全体の月別売上		範囲					3
			駐車場・洗車場全体の年別売上		範囲					3
			つり銭の残金（どの硬貨が何枚残っているか）		現在					2
	道路監視（路面凍結）	道路（路面温度センサ）	特定の道路の現在の路面状況		現在					2
			特定の道路の特定日時の路面状況ログ		一時点					2
			ある地域内にある道路のうち、凍結している道路のIDと場所情報		現在					4
	橋梁監視	橋梁（振動センサ、監視カメラ）	特定の橋梁の現在の状況		現在					2
			特定の橋梁の特定日時の状況ログ		一時点					2
			ある地域内にある橋梁のうち、危険な状態にある橋梁のIDと場所情報		現在					4
	信号機の遠隔制御	信号機	制御対象となる信号機のセンサ値および交差点・横断歩道状況が分かる映像							1

	標識やガードレールにおける自動車の衝突通知	標識、ガードレール	衝突があった箇所のID、場所リスト							5
	車両の盗難防止（異常通報後の位置監視）	自動車、オートバイ	特定の車両の現在の位置		現在					2
小売・流通	自動販売機の遠隔監視、保守支援	自動販売機	特定の自動販売機の現在の状態（各商品の残存数、つり銭、温度）		現在					2
			ある地域に存在する自動販売機の中で、メンテの必要がある自動販売機のIDと場所とメンテ内容のリスト		現在					4
	鮮度管理	温度計など（トラック）	特定のトラックの現在の温度		現在					2
			特定のトラックの特定の時間の温度		一時点					2
			特定のトラックの指定した期間での指定した間隔での温度変化		範囲					2
			ある条件を満たすトラックの異常値があったもののID、期間、温度のリスト		範囲					4
製造	集中管理された工場でのメンテナンス	製造装置、製造機械、建設機械	特定の装置の現在の状況（ON／OFF、温度、位置など）		現在					2
			特定の装置の指定した期間での指定した間隔での状況		範囲					2
			ある条件を満たす装置の中で異常値があったもののID、期間、値のリスト							4
	温度監視	炉、薬品庫	特定の炉、薬品庫の現在の温度		現在					2
			特定の炉、薬品庫の指定した時間、指定した間隔での温度リスト		範囲					2
			ある条件を満たす炉、薬品庫の中で異常値があったもののID、期間、温度のリスト							4
	異常通報	汚水処理装置など	ある条件を満たす装置の中で異常値があったもののIDリスト							4
	土壌地下水汚染のリアルタイム監視	工場・廃棄物処分施設	特定の施設の現在の状態（汚染物質の濃度など）		現在					2
			特定の施設の特定の時間、特定の間隔での状態値リスト		範囲					2
			ある条件を満たす施設の中で異常値があったもののIDリスト							4
	PL対策（使用回数、使用時間等）	製品	ある製品の使用開始から現在までの使用回数、使用時間		現在					2
			ある製品の使用開始から特定の時間までの使用回数、使用時間		一時点					2
家庭	家電等の遠隔監視・制御	照明、エアコン、電気錠、暖房器、給湯器、風呂、シャッター、録画機	ある家電の現在の状態（開／閉、温度など）		現在					2
	独居老人の安否確認（見守りポット、緊急通報）	ポット、仏壇、灯明	特定の器具からの情報		現在					2
	電気・ガスメーターの遠隔検針	電気・ガスメーター	特定のメーターからの現在の値		現在					2
			特定のメーターからのある時刻での値		一時点					2
			異常値を示しているメーターのIDリスト							4
	物品探し	貴重品	ある条件を満たす貴重品の現在の位置 「先週購入した赤いバッグが今どこにあるか」		現在					4
	代替機器利用	テレビ、ディスプレイ、プリンタ、電話	ユーザに近接した代替機器のNWアドレス 「現在Aさんの近くにある、使用中でない電話（に接続したい）」		現在					4
	ペット管理（迷いペット対策、盗難防止）	ペット（位置情報）	特定のペットの現在値、映像		現在					2
			特定のペットのある期間の行動履歴		範囲					2
環境・自然	廃棄物管理（産廃業者による不法投棄の防止）	不法投棄場所（センサ）	ある期間内に廃棄物が投棄されたときの映像リスト		範囲					4
	動植物の生態監視（位置情報）	野生生物	特定の生物の現在値、映像		現在					2
			特定の生物のある期間の行動履歴		範囲					2
金融・保険	ATMの破壊窃盗対策（異常通知後の位置通知）	無人ATM	特定のATMの現在値（異常発生後）		現在					2

建設・土木	工事現場の工事状況監視、危険予知	工事現場（雨量センサ、風速センサ）	特定の工事現場の現在の状態値リスト（雨量、風速など）		現在					2
			異常値を示しているメーターのIDリストと異常値							4
その他	貴重品管理（物品の開封の通知・証跡）	貴重品	特定の貴重品の現在の場所		現在					2
			特定の貴重品の開封							5
	電光掲示板への表示	電光掲示板	ある地域にある電光掲示板のIDリスト							4

表 1 より，アプリケーションによって多様な検索条件が用いられることが分かる．これら検索条件を「＜ID＞指定型」「＜ID，時間＞指定型」「＜位置，時間＞指定型」「その他の多次元指定型」「継続クエリ指定型（イベントドリブン型のデータ要求）」の 5 パターンに分類し，表 1 内での出現回数をカウントすると，表 2 のようになる．表 2 より，＜ID，時間＞の出現回数が高いことがわかる．表 1 内での出現回数は必ずしも当該検索の実施頻度に比例するわけではないが，蓄電池管理アプリケーションと同様に，多様な M2M アプリケーション全体の傾向としても，＜ID，時間＞による検索頻度は高いものと推測される．なお，「継続クエリ指定型」とは，M2M アプリケーションが何らかの検索条件を予め検索機能に登録しておき，当該検索条件を満たす M2M データが蓄積されたらすぐに当該 M2M アプリケーションに通知されるタイプの検索である．例えば表 1 内の運送・物流業界における「危険物配送トラックの事故速報」アプリケーションであれば，温度が異常な高さになったり，エアバッグが ON になったりしたデータを検索する検索条件を予め検索機能に登録しておき，該当するデータが発生した場合にすぐ通知を受けることで，当該アプリケーションは事故発生を迅速に検知することが可能となる．

表 2　各検索パターンの出現回数

検索パターン	出現回数
1:＜ID＞指定型	4
2:＜ID，時間＞指定型	101
3:＜位置，時間＞指定型	18
4:その他の多次元指定型	32
5:継続クエリ指定型	22

2.3 本論文で着目する検索要求

前述したように，水平統合型プラットフォームにおいては，大量のデータを蓄積しながら，大量の検索処理を実行するため，挿入性能，検索性能ともに高速性とスケール性が求められる．2.1，2.2 より，各 M2M アプリケーションは種類によって様々な検索条件により検索を行うため，各検索パターンに対し，高速性とスケール性を合わせ持つ検索機能を実現する必要がある．特に＜ID，時間＞の 2 次元の検索条件（検索パターン 2）により検索が実施される頻度が高いと予想されるが，やはり各検索条件が用いられる頻度を正確に予測すること

はできず，また頻度は動的に変化すると考えられるため，どのような検索をどれほど効率的に処理可能とすべきかを事前に決めることは困難である．従って，入力データや検索要求の変化にもある程度柔軟に対応可能な負荷追従性が求められると考えられる．

検索パターン 1，2 は ID を指定する検索であり，1 次元検索のための B-Tree やハッシュテーブルによるインデキシング技術が適用可能である．これらの技術は古くから高速性やスケール性，負荷追従性に関する技術改良がなされ，実用化が進められている [18] [19]．水平統合型プラットフォームにおいても，これらの技術を適用することにより検索パターン 1，2 に対する検索機能は実現可能であろう．また検索パターン 3 は位置や時間を指定する検索であり，こちらは時空間データベースの分野で多くの検討がなされている [20] [21]．水平統合型プラットフォームにおいても，やはりこれらの技術を適用することで実現可能であると考えられる．一方，検索パターン 4 については，多様な M2M データに対する多様な（例えばユーザやアプリケーションによって必要とされるデータが異なる）多次元検索というこれまでにない検索要求であり，従来技術では必ずしも十分な高速性，スケール性，負荷追従性を得ることができない．従来の 1 次元インデキシング技術であるハッシュテーブルや B-Tree を用いて当該多次元検索を効率化する場合，多次元検索条件（AND 条件）に含まれる 1 次元の条件で検索した検索結果（中間解）に対し，他の次元の条件を満たすかどうかを検査する手法（フィルタリング），または，各次元の条件で検索し，それぞれの中間解を照らし合わせ，全ての中間解に出てくる検索結果を抽出する手法（マージ），あるいはそれらを組み合わせた手法が用いられる．しかしながらいずれの手法においても，中間解の量が大規模となる場合には，フィルタリングやマージに要する処理コストが大きくなってしまうという問題がある．

そこで，本論文では，検索パターン 4 の多次元検索に対し，高速性とスケール性，負荷追従性をもつ検索機能を実現する新しいインデキシング技術について提案する．一口に検索といっても，完全一致検索（= キーワード検索），範囲検索，近傍検索，正規表現検索，ユーザ定義検索など様々な検索タスクが存在するが，本論文では特に，基本的かつ必要性が高いと考えられる完全一致検索および範囲検索に焦点を当てることとした．すなわち，多次元完全一致検索と多次元範囲検索が本論文での検討対象である．実際，表 1 内に出現する検索パターン 4 の多次元検索は，多次元完全一致検索または多次元範囲検索のどちらかである．以下，まず第 3 章では，多次元完全一致検索のために新たに提案するインデキシング技術「準候補キーインデキシング」について述べ，第 4 章では，多次元範囲検索のための新たなインデキシング技術「UBI-Tree」について述べる．

なお，検索パターン 5 については，従来 RDB におけるトリガ機能やアプリケーションレベルで実現されてきた他，近年では Publish / Subscribe システム [22]やストリームデータベース [23] [24]の研究が盛んに行われている．水平統合型プラットフォームにおいても，これらの知見を活かすことができると考えられるが，本論文で提案する準候補キーイ

ンデキシングや UBI-Tree を用いて更なる高速化，スケール性，負荷追従性を持たせることが可能である．UBI-Tree は，通常，蓄積データをインデキシングし，検索条件による検索を高速化するためのものであるが，検索条件をインデキシングし，データによる検索の高速化に用いることで，検索パターン 5 のような継続クエリ（Continuous Query）にも応用可能である [25].

2.4 M2M アプリケーションの検索要求まとめ

本章では，M2M アプリケーションの検索要求について検討・整理するため，まず例題として市川研究室で検討が進められている蓄電池管理アプリケーションの検索要求について考察し，これに倣いさらに 14 分野，計 58 種の M2M アプリケーションについて検索要求を検討・整理した．この結果，M2M アプリケーションの検索要求には大きく 5 つの検索パターンが存在することが判明した．さらに 5 つの検索パターンの中で，「その他の多次元検索」と分類した検索パターン，すなわち多様な M2M データに対する多次元検索が，これまでにない検索要求であるため，本論文ではさらに，当該検索を効率化するインデキシング技術を提案することを述べた．なお，「その他の多次元検索」とは，「ID」を検索条件に含む多次元検索や，「位置」と「時間」を検索条件に含む多次元検索以外の多次元検索であり，例えばセンサ値そのもの，あるいはその他属性情報等を検索条件に指定した多次元検索などである．また当該「その他の多次元検索」には多次元完全一致検索と多次元範囲検索が含まれる．

第 3 章
多次元完全一致検索のための
インデキシング技術

3.1 M2M データに対する完全一致検索の特徴と提案するインデキシング技術概要

第 2 章で述べたように，多次元検索は多くの M2M アプリケーションにとって有用である．特に，M2M データの多次元完全一致検索が実現できれば，RFID タグやセンサの普及によりネットワーク上に溢れつつある実世界上の事物（**リソース**）を，ユーザの必要に応じて自在に探し出し，利用することが可能となる．例えば，「現在 A さんの近くにある，使用中でない電話に接続したい」であるとか，「先週購入した赤いバッグが今どこにあるのか探して欲しい」といった要求に対しても，リソースの性質や状態などを表す属性情報を検索キーとして利用することが可能となる．RFID タグやセンサには，一意の ID が割り当てられていると想定されるため，検索キーとして ID を用いることもできるが，ユーザが個々の ID を覚えていると考えるのは非現実的である．不特定多数の中から，何らかの条件を満たすリソースを探す場合も多い．従って，意味のない ID ではなく，リソースを直接表現する属性情報を用いて検索できることが重要となる（ID を指定する場合は検索パターン 1 や 2 と同様の方法で実現可能である）．

前述した通り，水平統合型プラットフォーム上の検索機能には，高速性に加えてスケール性，負荷追従性が求められる．例えば SCM（Supply Chain Management）アプリケーションにおいて，属性情報による企業横断的な検索を実現する場合，リソース数は現在の Web ページ数よりもさらに多くなるであろう．また現在，多くの Web 検索エンジンでは，情報の更新時間を高速にするために様々な工夫が行われているが，全リソースの更新をリアルタイムに行うまでには至っていない．これを実現するためには，より分散化・並列化した構成を採ることが必要であろう．

分散化・並列化させる場合の構成としては，大きく階層型と P2P 型（水平分散型）に分

けることができる．階層型としてはDNS（Domain Name System）やePCグローバル [26]のONS（Object Name Service）等が代表的であり，スケール性のある構造として実績がある．しかしながら，階層構造では，その構造に合わせた検索方法を採らなければ効率的に処理することが出来ない．DNSであればサブネット，ONSならば製造者や商品種別により検索を行うからこそ階層化のメリットがあると言える．一方，M2Mデータの多次元完全一致検索において想定されるのは，IDだけでなく，登録されている情報の全てが検索キーとなる可能性があり，検索キーを特定できない場合であり，そうした場合適切な階層構造を採ることは不可能である．そこで，よりフラットな構造，すなわち互いに対等な関係で結ばれたP2P型での効率的な検索を行う方針を採用する．近年注目を浴びているP2P検索システムは，スケール性や負荷追従性の面で優れており，本検索機能を実現する技術として有望と考えられる [27].

P2P検索システムは,インデックスの分割方法により大きく2つのタイプに分けられる．1つはインデックスをリソース単位で分割するもの，もう1つはキーワード単位で分割するものである．クエリルーティングの方式から，前者はフラッディングベースのP2P検索システム，後者はDHT（Distributed Hash Table）ベースのP2P検索システムと呼ばれることもある．DHTとは，ハッシュテーブルを複数のピア（ホスト）で分散管理する技術であり，CAN [28]やChord [29]，Pastry [30]，Tapestry [31]などが代表的である．ファイル共有など，多くの複製が存在している場合にはフラッディングが有効だが，複製の存在を前提としない場合にはDHTが適している．実世界上のリソースは，複製されるものではなく，むしろ個々が明確に識別されるべきである．従って実世界上のリソースの検索には，フラッディングよりもDHTが適していると考えられる．しかしながら，DHTベースのP2P検索システムでは，キーワード単位でインデックスが分割されているため，複数の属性情報を含むような複雑な検索クエリになるほど多くのピア間通信が必要となり，検索処理に要する通信量が大きくなってしまうという問題点が指摘されている [32] [33].

そこで，検索クエリ 1 つあたりに要する通信量の上限を低く抑えることを可能とする，新しいインデキシング技術「準候補キーインデキシング」を提案する．準候補キーインデキシングは，従来の 1 次元インデキシング技術であるハッシュテーブルをベースとするが，AND条件（AND演算子で結んだ論理式による検索条件．以後，AND演算子を∧で表現する）を用いた場合にも高速に検索結果を返却できるよう，AND条件とそれに該当する検索結果リソース集合も予めインデックスに入れておく手法である．準候補キーインデキシングは，検索結果リソース数がある一定数（T個）以下のAND条件のみをインデキシングすることで，複数の属性情報によるAND条件の解決に要する通信量の上限値を抑えることを可能とする．

図 5 に，多次元完全一致検索（キーワード検索）技術における準候補キーインデキシング技術の位置付けを示す．図 5 に示すように，扱うデータ量が小さい多次元完全一致検索

は，デスクトップ検索や LAN 上のドキュメント検索において用いられ，全文検索技術等が活用され既に多くの製品が存在している．また扱うデータ量が大きく，かつ各データの重要度の偏りが大きい場合の多次元完全一致検索は Web 検索において用いられる．1.1.4 節で述べたように，Web データはあるユーザにとって価値のある Web データは多くのユーザにとっても有用である．すなわち，各データの重要度の偏りが大きく，このため使用される検索条件の偏りも大きい．このような場合，例えば検索結果をキャッシュすることで検索を効率化する手法 [17]が有効となる．一方，M2M データは使う側によって価値が変わり，各データの重要度の偏りが小さい．準候補キーインデキシングは，これまでのインデキシング技術とは異なり，検索結果リソース集合が T 個以下となる検索条件であればどのような検索条件でも効率化できることから，M2M 検索における多次元完全一致検索に適していると言える．

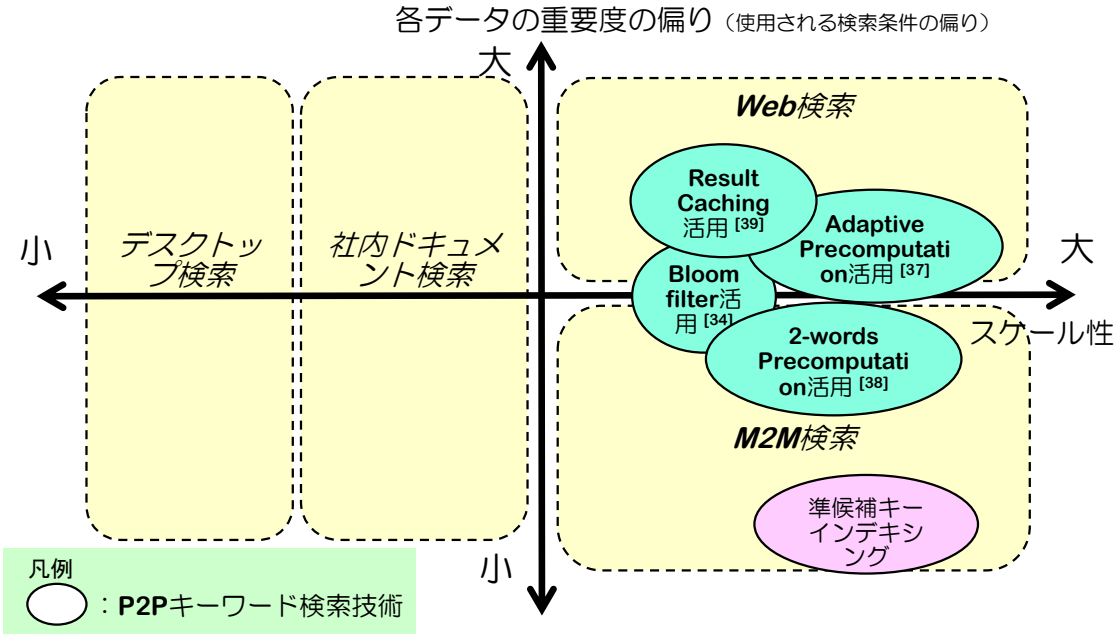


図 5（再掲）多次元完全一致検索技術における準候補キーインデキシング技術の位置づけ

以降，3.2 では，問題とその解決に対する考え方について述べる．3.3 では，準候補キーインデキシング技術の詳細と共に，システムの概要について説明する．3.4 では，当該システムについての評価，考察を述べる．3.5 では，関連研究について述べる．

3.2 問題とその解決に向けてのアプローチ

3.2.1 DHT ベース P2P 検索システムにおける問題

例として，X 社製のネットワークカメラを検索するために，「X 社製」と「ネットワークカメラ」の 2 つの属性情報から成る AND 条件で検索することを考える. DHT においては，「X 社製」という条件に結びつけられたリソースの情報（「X 社製」という属性情報を含む M2M データ）を保持するピア（X 社製担当ピアと呼ぶ）と，「ネットワークカメラ」という条件に結び付けられたリソースの情報を保持するピア（ネットワークカメラ担当ピア）は，各々の条件に対するハッシュ値のランダム性により通常異なる．このため，その AND 条件に該当するリソースの情報を得るためには，X 社製担当ピアで得られる「X 社製」のリソース集合と，ネットワークカメラ担当ピアで得られる「ネットワークカメラ」のリソース集合との積をとる集合演算を行う必要がある．このため，X 社製担当ピアとネットワークカメラ担当ピアとの間で通信を行い，一方の保持するリソース集合の情報（これを**中間解**と呼ぶ）を他方に送らなければならない. 積をとった結果，該当するリソースがたとえ数個であったとしても，それを求めるために膨大な通信が必要と成り得るのである．

この中間解の通信は，フラッディングベースの P2P 検索システムには存在しない，DHT 特有のものである．これにより，AND 条件による検索を行う場合，DHT は検索処理に多くの通信を要するという問題を抱えている．

3.2.2 インデックスの絞り込み

検索に要する通信量には，何を検索キーとするかにより，ばらつきがある．例えば，A さんの赤いバッグを探す場合に，「赤い」という条件で検索した場合と，A さんの氏名を条件とした場合とでは，検索結果数が大きく違う．前者の場合，赤いリソースは全て適合してしまうために検索結果は膨大となり，後者の場合，A さんに関連したリソースだけになるため検索結果をある程度絞ることができ，通信量も少なく済む. 検索に要する通信量を抑えるためには，後者のように絞り込みに効果的な検索キーのみを用いることが有効であろう. そもそも検索結果が多過ぎる場合，ユーザはその結果をそのまま利用することができず，検索条件を強めて再度検索しなければならない. つまり，そのような検索に膨大な計算資源やネットワーク資源を使用して答えたとしても，ユーザにとっては意味がない. 逆に検索結果を絞り切れないクエリに対しては答えないとすることで，計算／ネットワーク資源を有効利用できるとともに，ユーザを効果的な検索に導くことができるようになると考える．

以上より，検索結果が一定数以下に絞られる，有用な検索キーに対してのみ，インデックスを構築しておくこととする．

3.2.3 AND 条件のインデキシング

しかしながら，前節のようにしただけでは，ユーザは 1 つの属性情報のみで十分に絞られる検索キーを用いなければならず，ユーザの利便性に問題が生じる．先ほどの例において，「X 社製」と「ネットワークカメラ」とは，それぞれ単独では該当するリソースが多量にあったとしても，その AND 条件ならば十分に絞られる場合がある．そうした検索結果は，ユーザにとって有用であり，検索システムは答えられるべきである．そこで，このように 1 つの属性情報だけでは検索結果が多量でも，複数の属性情報を組み合わせることにより有用となる検索キーについては，その組み合わせそのものを検索キーとしてインデックスに入れる．これにより，ユーザが検索できる検索キーの範囲を広げ，ユーザの利便性を向上させる．

以上より，AND 条件であっても，検索結果が一定数以下に絞られる有用な検索キーであれば，インデックスに登録することとする．

3.2.4 シーケンシャル検索の利用

検索処理には大きく 2 つの方法がある．1 つは事前にインデックスを作成しておく方法，もう 1 つはインデックスを使わずにシーケンシャルにパターン照合する方法である．大量のデータを扱う場合，インデックスを利用することは必須であるが，データ量が少ない場合には，後者の方が簡単で高速な場合もある．

前節で検索結果数が一定数以下に絞られる AND 条件をインデックスに登録する，と述べた．しかしながら，そうした AND 条件は大量に存在するため，通信量の問題に代わりインデックス量の爆発という新たな問題が発生する. 例えば,「X 社製 ∧ ネットワークカメラ」という AND 条件が一定数以下に絞られる場合，「X 社製 ∧ ネットワークカメラ ∧ ○年○月製造」なども，一定数以下に絞られる AND 条件である．

そこで，一定数以下に絞られる全ての AND 条件をインデックスに登録するのではなく，１つでも条件が欠けると一定数以下に限定できなくなる AND 条件のみをインデックスに登録することとする．そうすることで，インデックス量の爆発を回避する．

「X 社製 ∧ ネットワークカメラ ∧ ○年○月製造」という AND 条件で検索された場合には, その中に含まれる「X 社製 ∧ ネットワークカメラ」という条件で検索することでも，

一定数以下に検索結果を絞ることができる．後は「○年○月製造」かどうか，一定数以下の検索結果 1 つ 1 つについてシーケンシャルに検査すれば良い．検査対象が一定数以下であることが保障されているので，処理負荷も一定量以下となる．

3.2.5 準候補キーの導入

以上の考察より，「**準候補キー**[1]」という概念を導入する．次の 2 つの条件を満たす検索キーを準候補キーと定義する．

●条件 1　該当するリソース数が T 個以下に絞られる

●条件 2　（AND 条件の場合）条件が 1 つでも欠けると T 個以下に限定できなくなる

T を**閾値**と呼ぶ．この値により，当該の検索キーが有用かどうかを分ける．この準候補キーをインデックスとして登録しておき，検索時には検索条件に含まれる準候補キーを見つけ出し，そこに登録されたリソース群をシーケンシャルに検査することで，中間解の通信量を抑えながら，ユーザに利用しやすい検索システムの実現を目指す．

3.3 提案する準候補キーインデキシング技術を用いたシステム

3.3.1 システム概要

準候補キーインデキシング技術を適用した DHT ベース P2P 検索システムについて説明する．ただし，準候補キーインデキシング技術は，「登録時には M2M データ（リソースを示す属性情報群）から準候補キーを抽出しインデックスに登録し，検索時には検索条件から準候補キーを抽出しインデックスから検索する」アルゴリズムであり，ハッシュだけでなく B-Tree などのインデックス構造にも適用することができる．また，単体データベースでも

[1] リレーショナルデータベースの分野において，テーブル中のタプル（行）を一意に同定することができる属性，または属性の組で，かつ，そのうち属性が 1 つでも欠けると一意に同定できなくなるものを候補キーと呼ぶ．準候補キーはこれに準ずるものとして名付けたものである．

分散システムでも，同様に適用することが可能である．

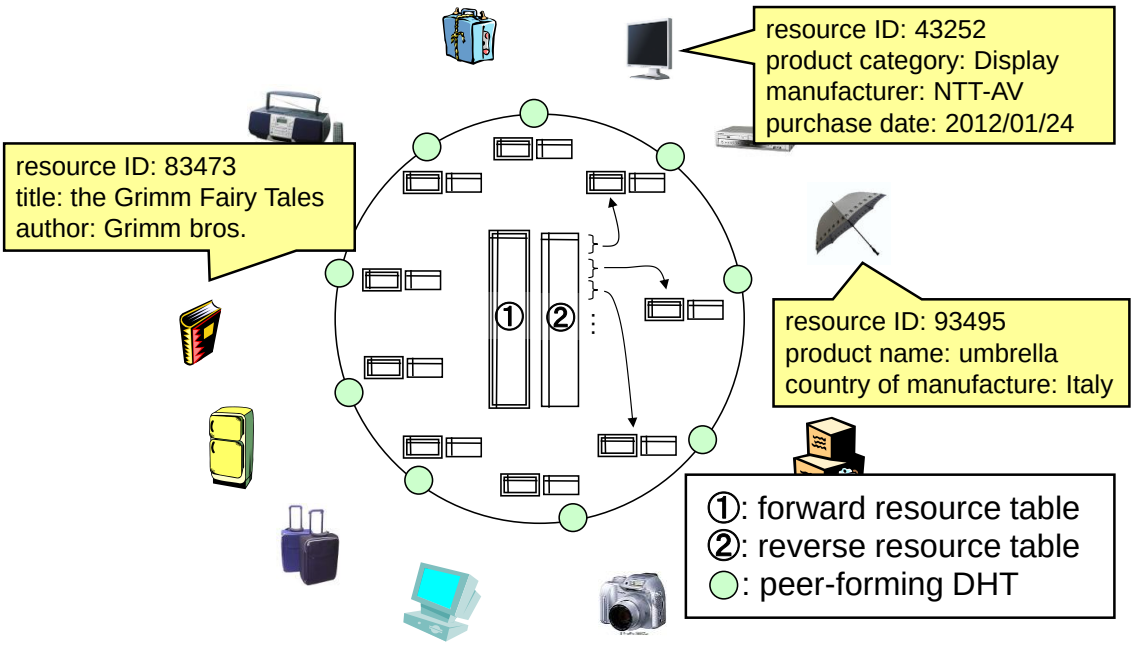


図 8　準候補キーインデキシング技術を用いたシステムの概要

システムの概要を図 8 に示す．様々なリソースは，一意の ID を持ち，またそれぞれ様々な属性情報を持つことを前提とする．図 8 に示したように，こうしたリソース群の情報を保持する巨大なハッシュテーブル（以下，単に**表**と呼ぶ）を，複数のピアが DHT を形成し，分散管理する．本システムでは，2 種類の巨大な表を DHT により分散管理する．1 つはリソース ID をキーとするインデックスを持ち，値としてそのリソースの属性情報群を持つ表（これを**順リソース表**（forward resource table）と呼ぶ），もう 1 つはリソースの属性情報群から得られた準候補キーをキーとするインデックスを持ち，値としてその準候補キーに対応するリソース ID 集合を持つ表（これを**逆リソース表**（reverse resource table）と呼ぶ）である．リソース ID を検索キーとし，そのリソースの属性情報を知りたい場合には順リソース表を利用し，逆に属性情報を検索キーとし，それに該当するリソース ID を知りたい場合には逆リソース表を利用する．順リソース表，逆リソース表の例を図 9 に示す．逆リソース表には，各キーに対する値となる ID の数に閾値（T）が設けられており，それ以上 ID を保持することは許されない．準候補キーの条件 1 に反するためである．図 9 では，T=3 の場合を示した．各タプル（行）には「溢れフラグ（overflow flag）」のためのデータ領域が用意されており，T 個を超える ID が登録されようとした場合には，この溢れフラグを立てておく．この溢れフラグが立っているキーは，準候補キーではないことを意味している．

また同様に，各タプルには，「再登録中フラグ（under reregistration flag)」のためのデータ領域が用意されており，データの一貫性を保つために用いられる．これについては後述する．

以下，これら 2 つの表を使い，検索システムの重要な機能である登録・削除・更新・検索それぞれの実現方法について述べる．

resource ID	attribute
74839	product name = color TV, date manufactured = 2014/5/7 location of manufacture = abc factory
30991	product name = color TV, date manufactured = 2014/5/7 country of manufacture = Japan, weight = 12.4 kg

overflow flag　　under reregistration flag

attribute			resource ID		
product name = color TV	0	0	74839	30991	69022
date manufactured = 2014/5/7	1	0			
location of manufacture = abc factory	0	0	74839	20478	
country of manufacture = Japan	1	0			
weight = 12.4 kg	0	0	30991	83165	24751
date manufactured = 2014/5/7 ∧ country of manufacture = Japan	0	0	30991	03211	

図 9　順リソース表（上）と逆リソース表（下）

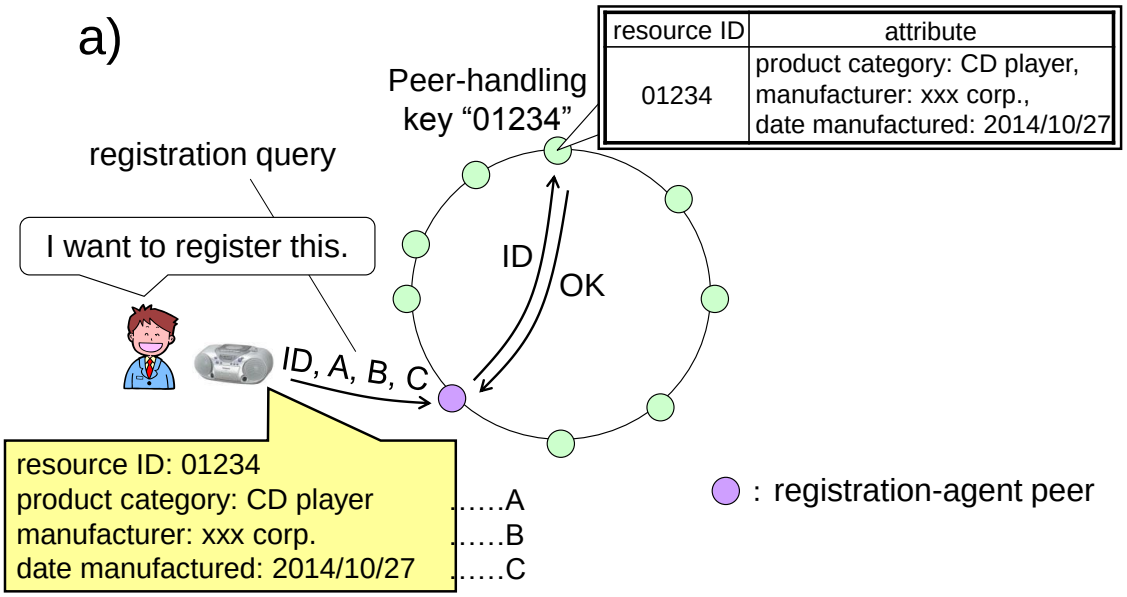
3.3.2 リソース情報登録アルゴリズム

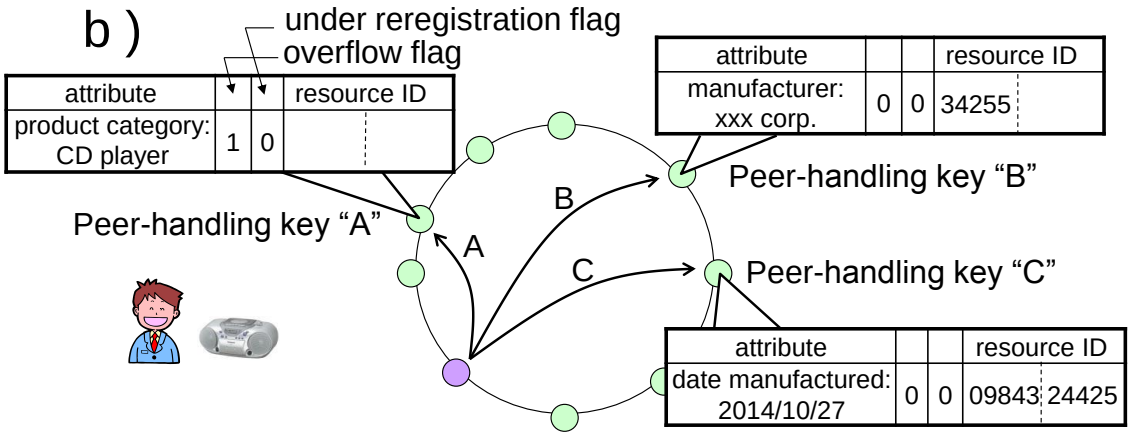
3.3.2.1 新しいリソース情報の登録

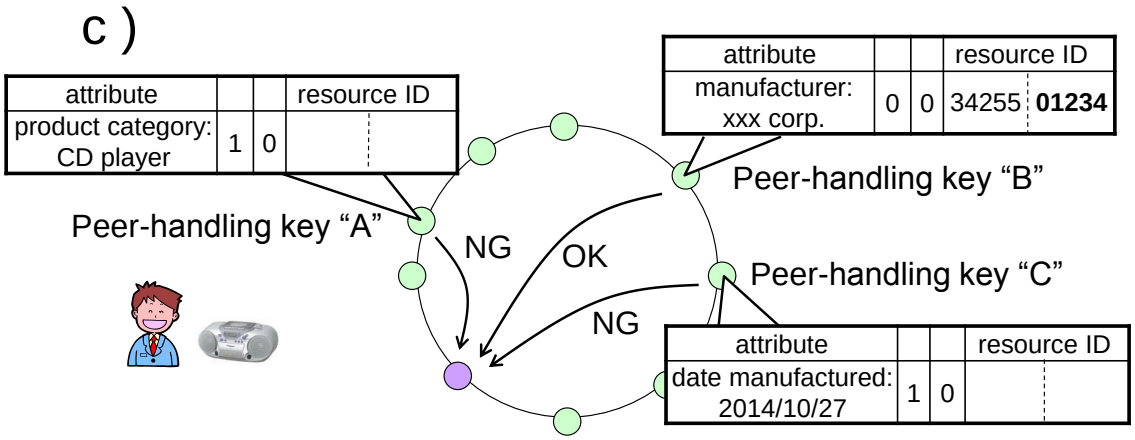
リソースを登録する際には，2 つの表に登録を行う．登録したいリソースのリソース ID とその属性情報群を受け取ったら，まず順リソース表に登録し，次に逆リソース表に登録する．以下，具体例を使ってその処理方法を説明する．

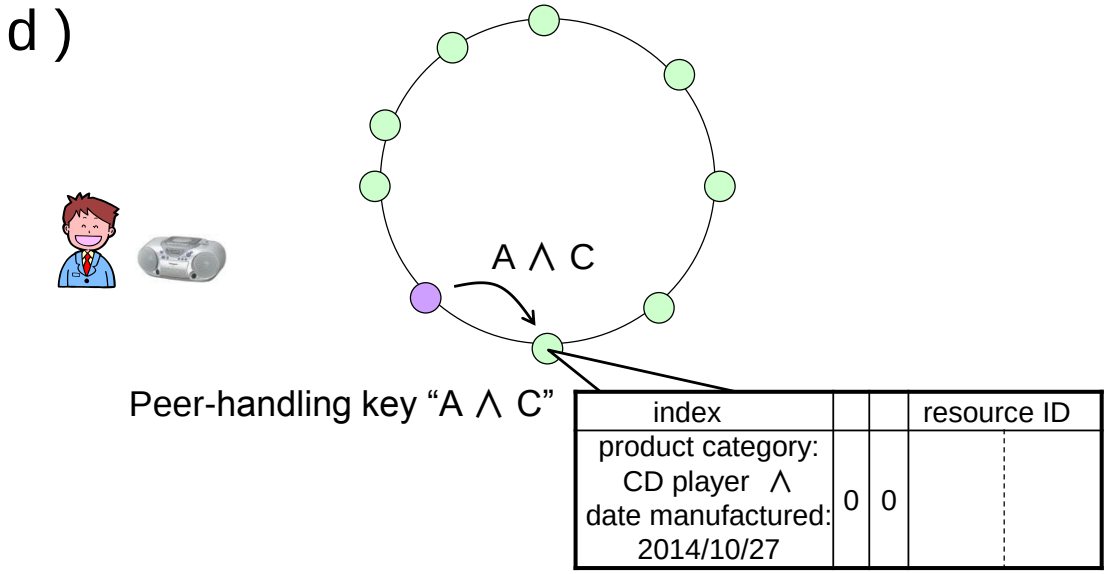
図 10 に示すように, ID が 01234 であり，product category : CD player，manufacturer :

xxx corp.，date manufactured：2014/10/27 という 3 つの属性情報を持つリソースを登録することを考える．以下，簡単のため，3 つの属性情報をそれぞれ A，B，C と表記する．ユーザは，DHT を構成するピアの 1 つに登録要求クエリを送信する．このピアを**登録代行ピア**（registration-agent peer）と呼ぶ．登録代行ピアはまず，順リソース表への登録を行う（図 10-a）．順リソース表への登録は，DHT における通常のデータ登録手順と同様である．逆リソース表への登録は，リソースの属性情報群に含まれる，準候補キーのみを登録する．準候補キーと成りえるのは，属性情報の任意の組合せから構成可能な検索キー全てであり，ここでは A，B，C，A ∧ B，A ∧ C，B ∧ C，A ∧ B ∧ C の 7 つである．登録代行ピアは，まず，このうち 1 要素から成る，A，B，C の 3 つについて，逆リソース表への登録を試行する（図 10-b）．該当するタプルに，既に T 個以上のリソースが登録されていた場合には，準候補キーの条件 1 を満たさないために登録が許されず，そうでない場合には，準候補キーであるために登録できる．この例では，T=2 であるとし，B への登録は成功するが，A と C のタプルには既に 2 つ以上のリソースが登録されており，登録できなかったとする（図 10-c）．これは，B は準候補キーであったが，A，C は準候補キーではなかったことを意味する．C のタプルには元々 2 つのリソース(ID:09843 と ID:24425 のリソース）が登録されており，今回の新しいリソースの登録により閾値を越える．このように閾値を越えて溢れた場合には，そのタプルの溢れフラグを立てておく．




b)
under reregistration flag
overflow flag
attribute
resource ID
product category: CD player
1
0
Peer-handling key “A”
attribute
resource ID
manufacturer: xxx corp.
0
0
34255
Peer-handling key “B”
Peer-handling key “C”
A
B
C
attribute
resource ID
date manufactured: 2014/10/27
0
0
09843
24425


c)
attribute
resource ID
product category: CD player
1
0
Peer-handling key “A”
attribute
resource ID
manufacturer: xxx corp.
0
0
34255
01234
Peer-handling key “B”
Peer-handling key “C”
NG
OK
NG
attribute
resource ID
date manufactured: 2014/10/27
1
0


d)
A ∧ C
Peer-handling key “A ∧ C”
index
resource ID
product category: CD player ∧ date manufactured: 2014/10/27
0
0

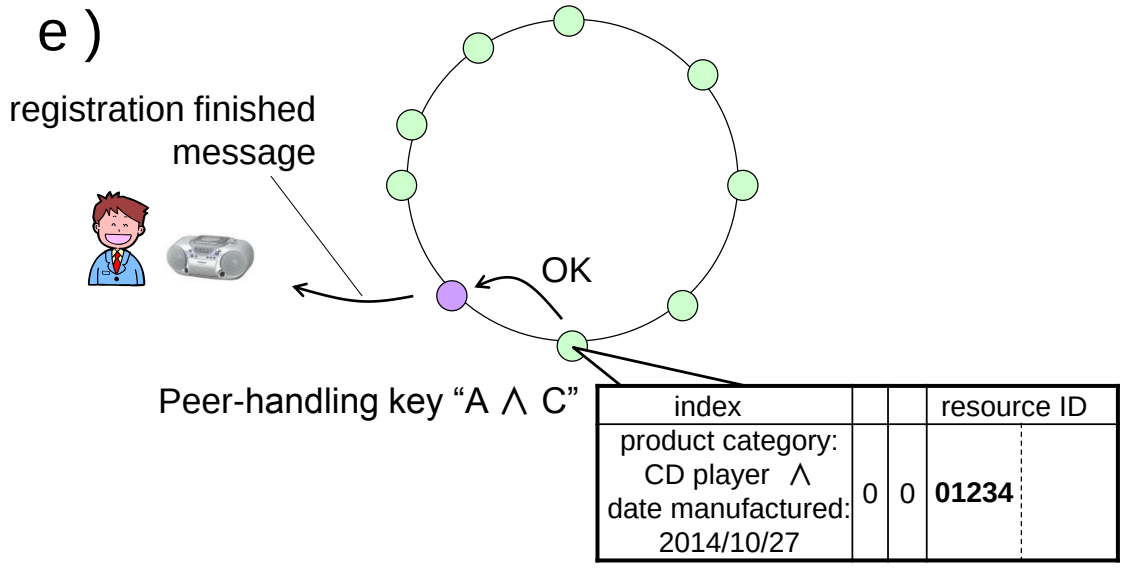


図 10　登録処理

ここまでで，先に挙げた 7 つの候補のうち，1 要素から成る 3 つの候補については準候補キーであるか否かの判別を完了した．これらの結果は，次に判別を行う 2 要素から成る候補の判別に役立つ．例えば A ∧ B は，B が準候補キーであるため，A という条件が欠けても該当リソースは 2 個以内であることが保障される．すなわち，準候補キーであるための条件 2 に反する．B ∧ C についても同様である．一方，準候補キーでなかった A，C を要素とする候補 A ∧ C については，準候補キーであるための条件 2 は満たすが，条件 1 については不明であるため，実際に登録を試みる必要がある（図 10-d）．残った候補 A ∧ B ∧ C であるが，これも先と同様に B を含むため，準候補キーでないことはすぐに分かる．全ての候補について，要素数の少ないものからこのように篩（ふるい）にかけていき，全ての候補について判別を終了したら，登録代行ピアからユーザへ登録完了メッセージを送る（図 10-e）．

登録代行ピアが行う処理を一般化すると，次のように表現することができる：

登録したいリソースが保持する M 個の属性情報について，1≦*i*≦M を満たす全ての整数 *i* に対し，*i*=1 から昇順に，*i* 個の要素からなる AND 条件のうち，それまでに登録できた AND 条件を包含しないものについて，逆リソース表への登録を試行する．

3.3.2.2 リソース情報の再登録

前述のように，図 10-b においては，既に C のインデックスには T と同じだけのリソースが登録されており，新たなリソースの登録により「溢れ」が発生するが，これは新たな登録により，準候補キーでなくなることを意味する．このようなとき，多くの場合，C と他の属性情報を AND 演算子で結びつけた，新しい準候補キーが発生する（例えば A ∧ C．今までは C だけで T 個以下に絞られたので準候補キーではなかった）．こうした新しい準候補キーをインデックスに登録するため，図 10-b の後，C のタプルに既に登録されていたリソースについて，システムへの再登録を行う．再登録は，C を担当するピア自身が登録代行ピアとなって行う．再登録すべきリソースの属性情報群は，順リソース表より取得する．C に既に登録されていたリソースの再登録が全て終わるまで，新しくできた準候補キーの「再登録中フラグ」を立てておく．再登録中には，新しい準候補キーに登録されるべきでありながら，まだ登録されていないリソースが存在している．その間に，この新しい準候補キーを検索条件とした検索が行われると，実際には検索結果に含まれるべきでありながら，まだ登録されていないために正しい答えを返すことができない．そこで再登録中フラグが立っている間は，この新しい準候補キーへの検索をブロックしておく．

3.3.3 リソース情報削除アルゴリズム

リソースを削除する際には，上述の 2 つの表から削除する．ユーザより削除したいリソースのリソース ID を受け取ったら，まず順リソース表から該当するリソースのタプルを探し，その属性情報群を取得した後，順リソース表からそのタプルを削除する．次に，逆リソース表から削除を行うが，手順は登録時のものと似通っている．属性情報群から構成される全ての検索キーのうち，まず 1 つの要素から成るキーについて，そのタプルから当該リソース ID を削除する．そのタプルが既に溢れていた場合には，そのタプルを保持する担当ピアから削除失敗メッセージ (NG) が返り，そのキーが準候補キーでなかったことが分かる．次に準候補キーでなかった 2 要素からなる AND 条件をキーとして，そのタプルからの削除を行う．リソースの属性情報数が M 個である場合，これを M 要素の AND 条件まで繰り返す．

登録時に，リソース情報から抽出した準候補キーのリストを順リソース表に書き込んでおき，再登録時や削除時にその情報を利用することで，再登録や削除の処理を効率化することも可能である．

3.3.4 リソース情報更新アルゴリズム

リソース情報の更新は，前記の削除処理，登録処理を順に行うことにより実現する．ユーザからリソース ID と新しい属性情報群を受け取ると，まず順リソース表より古い属性情報群を取得し，順リソース表と逆リソース表とから削除する．次に新しい属性情報群にて，新たにリソースを登録し直す．

3.3.5 リソース情報検索アルゴリズム

検索処理の大まかな流れとしては，まず逆リソース表を使って検索条件に含まれる準候補キーを見つけ，次にその準候補キーのタプルに登録されている T 個以下のリソースについて，順リソース表を使って検索条件全体を満たすかどうか検査を行う．検索条件と準候補キーとが同じものであれば検査の必要はないが，準候補キーが検索条件のサブセットであれば，検査は必須である．リソースの検査は，リソース ID から順リソース表を使ってそのリソースに関する属性情報全体を取得し，これが検索条件を満たしているかどうかを判断することにより行う．

検索条件から準候補キーを 1 つ見つける手法は，属性情報群から全ての準候補キーを見つけるために行った登録時の処理とは全く異なる．この手法では，次の 3 つの事実を利用する．

1) ある検索条件をキーとするタプルが存在し，溢れていない場合，その条件は準候補キーである
2) ある検索条件をキーとするタプルが溢れている場合，その条件は，該当リソース数を T 個以下に絞れない弱過ぎる条件である
3) ある検索条件をキーとするタプルが存在しない場合，その条件は，それに含まれる 1 つ以上の属性情報を外しても T 個以下（0 個を含む）に絞れる強過ぎる条件である

弱過ぎても強過ぎても準候補キーにはならない．ユーザからの検索条件が弱過ぎる場合，そこから準候補キーを見つけ出すことは不可能だが，強過ぎる場合には，その要素となっている属性情報の一部を条件から外し，弱めることで，元々の検索条件に含まれる準候補キーを見つけ出すことができる．以下，例として A ∧ B ∧ C ∧ D ∧ E を検索する場合の手順を説明する．以後，一時的に X ∧ Y を XY と略記する．

【手順 1】　ABCDE をキーとするタプルを調べる

i)　存在し，溢れていない場合（準候補キー発見），そこに登録されているリソース ID 集合が検索結果となる（検索処理終了）

ii)　溢れていた場合，検索条件は弱過ぎるため，ユーザに検索条件を強めるよう要求する

iii)　存在しない場合，検索条件は強過ぎることを意味する．手順 2 へ

【手順 2】　A，B，C，D，E の中からランダムに一つ選んで（例えば C とする）検索条件から外し，残りの属性情報から成る条件（ABDE）をキーとするタプルを調べる

i)　存在し，溢れていない場合（準候補キー発見），登録されているリソース ID の（条件 C に対する）検査を行い，検査に合格したものが検索結果となる

ii)　溢れていた場合，C は必ず条件に含めていなければならないことが判明（C を外すと条件が弱くなり過ぎる）．手順 3 へ

iii)　存在しない場合，C は検索条件に含める必要がないことが判明（C を外してもまだ条件は強過ぎる）．手順 4 へ

【手順 3】　A，B，D，E の中からランダムに一つ選んで検索条件から外し，残りの属性情報と C とから成る条件をキーとするタプルを調べる

【手順 4】　A，B，D，E の中からランダムに一つ選んで検索条件から外し，残りの属性情報から成る条件をキーとするタプルを調べる

以下同様に，検索条件に含まれる属性情報を 1 つずつ外して条件を弱め，準候補キーとなるかを調べる．弱めすぎた場合には元に戻し，他の属性情報を外す．属性情報を外す際，ランダムに選んでいるのは，負荷が特定のノードに集中するのを避けるためである．手順 2 以降，1 つの手順につき，1 つの属性情報について，検索条件に含めるべきか含めるべきでないかが決定する．このため，本検索方法では，本システムに登録された全リソース数に関わらず，最大 a+1 回（a は元々の検索条件に含まれていた属性情報数）の手順を行えば，検索条件に含まれる準候補キーの一つにたどり着くことができる．あるいは，検索条件に該当するリソースが存在しないことが判明する．

また検索条件が弱過ぎる場合には，手順 1 でそれが判明するため，ユーザに対し，即座に検索条件を強めるよう指示することができる．

OR や NOT を使った検索については，通信量の上限値を抑えることができない．そのため，システムを設計・実現するという観点からは，そうした検索については許可しない，という立場をとることが最もシンプルである．しかしながら，OR 検索を行いたい場合，例えば A or B という検索条件の場合には，A，B それぞれについて前記の検索方法で検索した後，その論理和をとれば良い．or で結ばれた条件それぞれが，準候補キーであれば検索は可能である．また，NOT 検索を行いたい場合，例えば not A という検索条件については，A が準候補キーであれば，まず A を検索条件として検索を行い，検索結果に含まれていないリソースを順リソース表から全て集めれば not A に該当するリソースを全て抽出できる．しかしながら，その該当数は膨大になりやすく，それに比例して処理負荷も膨大となる．not A に該当する数が小さい場合には，not A を属性情報として予め登録しておけば，not A が準候補キーとなり，検索することが可能である．

3.3.6 逆リソース表の定期的な更新

削除や更新により，今まで該当数が T 個以上であったキーが，T 個以下になり，再び準候補キーとなる場合がある．しかしながら，上述した削除／更新アルゴリズムでは，これに対応することができない．実際には溢れていないが，溢れフラグがたったままの「正しくないキー」が発生してしまう．

正しくないキーの発生を抑えるため，定期的に逆リソース表の再構築を行う．これは現在稼働中の逆リソース表とは別に，新たな逆リソース表を順リソース表から作成し直すものである．図 11 に示すように，一定期間（これを τ とする）ごとに，それまで検索に利用してきた逆リソース表を破棄し，再構築を終えた逆リソース表を利用し始める．それと同時に，また新たな逆リソース表の構築を始める．通常の登録・削除・更新クエリを受けた場合には，稼働中・準備中 2 つの逆リソース表に対して処理を行う．逆リソース表再構築のため，各ピアは，自身のもつ順リソース表に登録されている全リソースについて，各期間に一度ずつ，自身が登録代行ピアとなって新しい逆リソース表への登録を行う．ただし，ネットワーク機器など，リソースが常にオンラインであることを前提とするシステムにおいては，この定期的な登録の責任をリソースに課すことも可能である．

データの整合性を保つためには，逆リソース表の世代交代を行う前に，各ピアの順リソース表に登録されているリソース全てが新しい逆リソース表へ登録されることを保証しなければならない．このため，各ピアは逆リソース表の更新周期について同期をとる必要がある．この更新期間 τ が短い程，逆リソース表再構築のための負荷は高くなるが，「実際には溢れていないが，溢れフラグがたったまま」の正しくないキーを少なくすることができる．

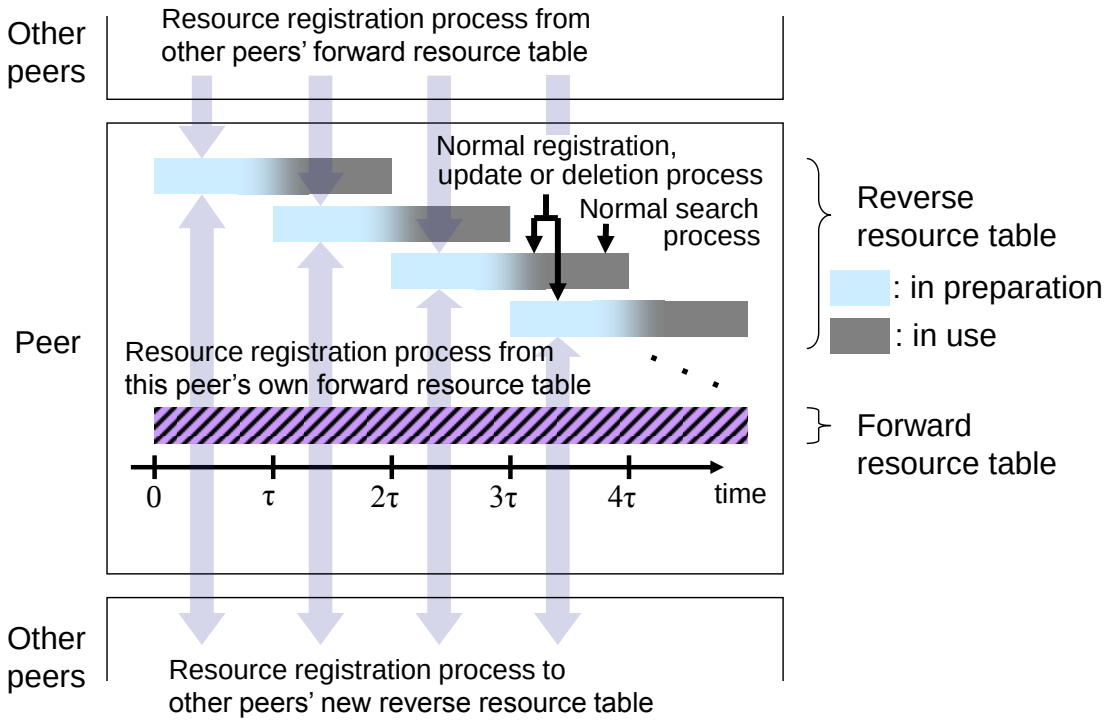


図 11　逆リソース表の更新処理

正しくないキーが存在すると，ユーザが準候補キーを含む検索条件で検索しても，「溢れている」，つまり十分に検索条件を絞り込めない検索条件であると見なされ，検索出来ない場合が生じる．ただし，これは検索クエリに含まれる準候補キーの全てがそうした正しくないキーであった場合であり，1 つでも正しいキーがあれば検索可能である．また検索できない場合も，ユーザが検索条件の追加を求められるだけであり，ユーザビリティを大きく損なうものではない．そもそも，一度でも閾値を越えたことがあるキーであれば，そのキーは絞り込む力が弱く，あまり有用でないと考えられるため，溢れフラグを立てたままにしていても，その影響は小さいと考えられる．更新期間 τ の長さを決める際には，そうした点を踏まえ，リソースの更新・削除頻度やユーザの要求，ネットワーク負荷やピアにかかる処理負荷のトレードオフにより，設定する必要がある． τ の設定方法については 3.4.1.3 でさらに議論する．

ただし，リソースは追加されるばかりで，削除，更新されることがないシステムにおいては，逆リソース表の更新は不要である．そうした場合には，逆リソース表は 1 つで良く，登録処理や検索処理は 1 つの逆リソース表に対して行う．

3.4 評価と考察

提案システムがスケール性を持ち，大規模システムへの適用が可能であるかどうか，評価を行う．具体的には，リソースの検索や登録に要する通信量，2 つの表を保持するのに必要となる記憶容量について，評価する．また，閾値 T の設定方法について議論する．

3.4.1 通信量についての評価

3.4.1.1 検索処理に要する通信量

A_i を 1 つの属性情報とすると，一般に，$A_1 \wedge A_2 \wedge \cdots \wedge A_a$ という検索条件を解決する際に必要となる通信は，**①準候補キーを探すための通信**と，準候補キーを見つけた後，準候補キー担当ピアが基点となって行う，準候補キーのタプルに登録されている**②T 個以下のリソースに対する検査に要する通信**，そして準候補キー担当ピアからユーザへの**③検査結果の送信**の 3 つである．議論を簡単にするため，ユーザ端末を含め，1 回あたりのピア間通信におけるホップ数は一定であるとし，（ピア間通信回数）×（パケットサイズ）を通信量として考えることにする．

① 準候補キーを探すための通信

検索処理アルゴリズムの節で述べたように，準候補キーは最大でも a+1 回の検索手順で探すことができる．各手順では，あるキーに対する担当ピアを探すための，オーバーレイネットワーク上でのルーティングに要する通信と，担当ピアからの返答のための通信が行われる．リソース ID やそのハッシュ値，あるいは溢れているかどうかなどを示すフラグを送るだけなので，1 つの通信パケットの大きさは数十～数百 byte である．これを m とする．またオーバーレイネットワーク上でのルーティングは DHT のアルゴリズムに依存するが，通常 $O(\log P)$回（P はピア数）のホップ数で実現される．この転送回数を p で表す．すると，準候補キーを探すための通信量は，最大で $m(a+1)(p+1)$となる．

② T 個以下のリソースに対する検査に要する通信

各リソースの検査では，そのリソース ID の担当ピアへ条件全体を送信し，担当ピアにおいて，そのリソース ID に関係付けられた属性情報群が条件全体を満たしているかを検査し，その結果を返す，という処理を行う．先ほどと同様，通信パケットには条件やハッシュ値が入るだけなのでサイズは m で近似する．検査対象のリソースは最大で T 個であるから，検査に要する通信は，最大で mTp である．

③ 検査結果の送信

検査結果はやはり T 個以下のリソース ID であり，リソース ID1 つを m で近似すると最大で mT の通信量を要する．

以上より，検索処理全体で要する通信量は最大 $m(a+1)(p+1)+m\mathrm{T}p+m\mathrm{T}=m(p+1)(a+1+\mathrm{T})$ となる．a はユーザが指定する属性情報数なのでそれほど大きくならないと考えられる．また m は数十～数百 byte，p は $\log P$ のオーダであり，T は設定可能であるから，検索処理に要する通信量は，リソース数に関わらず，またピア数に対してもほぼ変わらず，低く抑えることが可能であると言える．

3.4.1.2 登録処理に要する通信量

本アルゴリズムでは，通常の DHT よりも登録時に多くの手間をかけて索引を作っておくことにより，従来課題であった検索時の通信量を小さくしようとするものである．そのため，登録処理には従来より多くの通信を要する．そこで，擬似的に作成したリソース情報を使って本アルゴリズムの登録処理を模擬するシミュレーションを行い，登録処理に要する通信量を調べた．

擬似リソース情報として，N 個のリソース情報集合を作成した．このとき，1 リソースに関係付けられた属性情報数は平均 M の Poisson 分布に従うとし，属性情報の種類数は S 個，属性情報の出現頻度は Zipf 分布に従うとし，属性情報の出現の仕方は互いに独立とした．図 12 に示すように，DBLP [34]で公開されている 60 万件程度の文書情報 XML データを調査した結果，リソース数と属性情報の種類数は比例することが分かったため，S=N とした．

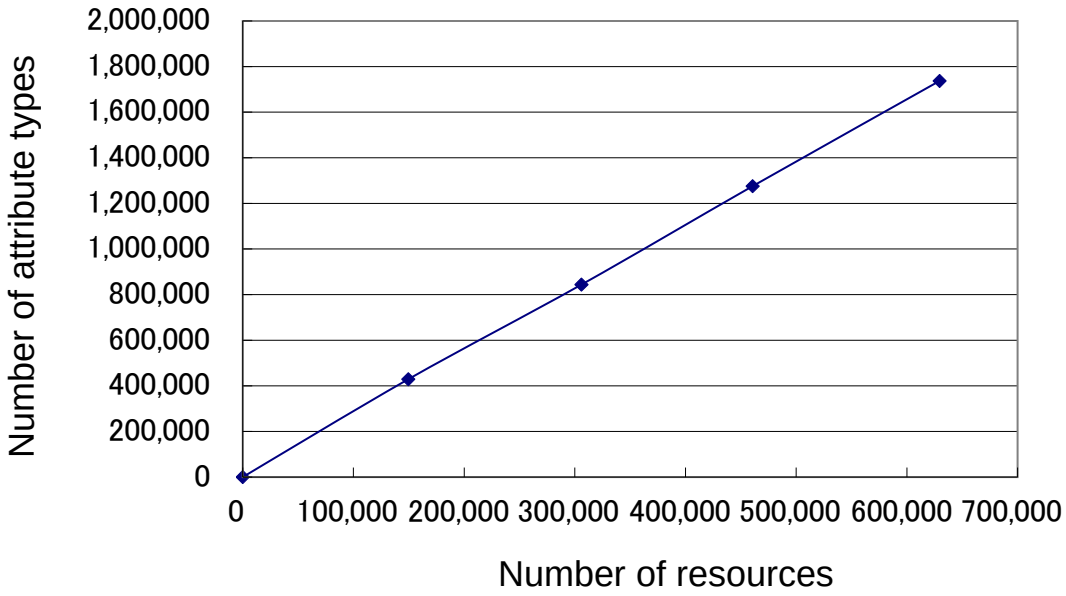


図 12　リソース数と属性情報種類数の関係

T=1000, M=20 とし，リソース数 N を様々に変化させた場合の，準候補キー数の変化と，1 リソースあたりの登録試行数をそれぞれ図 13，図 14 に示す．実験に際しては，3.3.3 で述べた再登録処理の効率化手法を取り入れた．登録試行 1 回あたりの通信量は *m(p+1)* と一定であるから，登録処理全体に要する通信量は，1 リソースあたりの登録試行数に比例すると言える．図 13 は，準候補キーの総数の変化と，それに含まれる準候補キーがいくつの属性情報から形成されているかの内訳を示している．図 13 より，リソース数が多くなる程，T を超えるキーが多くなり，より多くの要素を組み合わせた準候補キーが増加することが分かる．これは，リソース数が増加するに従い，登録試行数が増加してしまうことを意味している．しかしながら，図 14 より，リソース数が多くなる程，登録試行数の増加の仕方は緩やかになることが分かる．図 14 中の 11 点の実測値に対し単回帰分析を行い，回帰式を

$$y = c_0 \ln(N + c_1) + c_2 \tag{1}$$

とすると，決定係数は 0.9997 と高い値を示した．ただし，このとき N はリソース数，y は 1 リソースあたりの登録試行数，偏回帰係数はそれぞれ c_0=21.94622，c_1=3913.928，c_2=-166.6162 である．登録試行数がリソース数に対して *log* オーダで変化するため，登録処理全体に要する通信量についても，スケール性があると言える．

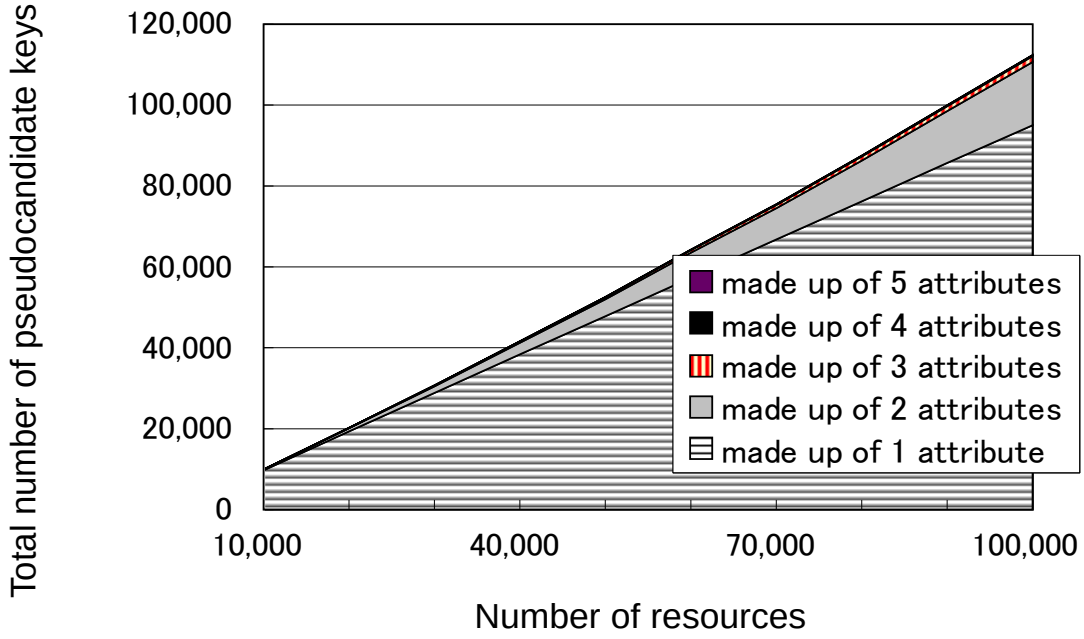


図 13　リソース数と準候補キー総数の関係

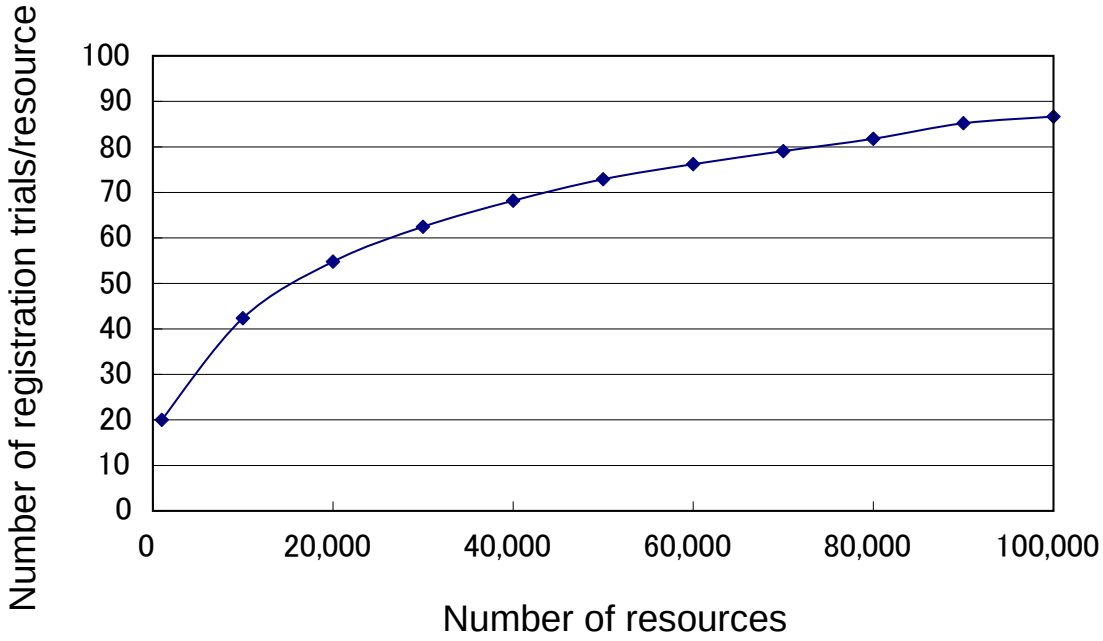


図 14　リソース数と登録試行数の関係

3.4.1.3 逆リソース表の更新に要する通信量

3.3.6 で述べたように，システム内のデータの整合性を正しく維持するためには，逆リソース表を定期的に再構築する必要がある．この処理にかかる負荷を見積もった．

「実際には溢れていないが，溢れフラグがたったまま」の正しくないキーは，そのキーを含む準候補キーに登録されたリソース ID 情報が削除された場合に発生し得る．更新処理は削除処理と登録処理を組み合わせたものであるから，削除処理だけでなく更新処理においても同様に正しくないキーが発生する可能性がある．リソースが 10 万件登録されている状態から，リソースが削除／更新された場合に，インデックス内の全キーうち何%のキーが正しくないキーとなるかを図 15 に示す．このとき，登録するリソース集合の分布には 3.4.1.2 と同様の条件を用い，M=20，T=1000 とした．また実験においては，簡単のため削除処理のみを行った．図 15 より，リソースが削除される程，正しくないキーが増加することが分かる．

各検索クエリには x 個の準候補キーが含まれ，インデックス内の全準候補キーは検索処理において等確率で利用されると仮定すると，「閾値以下に検索結果を絞り込める検索クエリにも関わらず，ユーザが絞り込み条件の追加をシステムから要求される」確率は，$(\text{正しくないキーの割合（\%）}/100)^x * 100$（%）で表すことが出来る．例えばこの確率を 0.1%まで許容し，x=2 である場合，正しくないキーは 3%程存在することが許されることになる．図 15 の場合であれば，登録されているリソースのうち，約 80%が削除，もしくは更新される周期で，逆リソース表の再構築を行えばよい，ということになる．この周期のうちに，3.4.1.2 で示した登録処理に要する通信量がネットワークへの負荷として発生することとなる．

今，登録処理，更新処理，削除処理がそれぞれ毎秒 Q 回ずつ行われ，常に N 個のリソースが登録されている状態を考える．登録処理 1 回の通信コストを C とおくと，削除処理もほぼ同じだけのコストがかかるため，更新処理，削除処理にはそれぞれ 2C，C の通信コストがかかると言える．すなわち，登録／更新／削除処理のために毎秒 4 QC だけの通信コストがかかることになる．一方，毎秒 2Q 回の更新処理，削除処理が行われるため，N 個のリソースのうち 80%が削除，もしくは更新されるのに要する時間は 0.8N/2Q 秒となる．これを逆リソース表の更新周期とすると，1 つの周期の中で N 回の登録処理が行われるため，それに必要な通信コストは毎秒(N/(0.8N/2Q))*C，すなわち 2.5QC となる．逆リソース表の更新のために費やされるこの通信コストにより，通常の登録／更新／削除処理だけの場合に比べ 1.625 倍の通信コストを要することとなる．もちろん，許容する正しくないキーの割合を高くすれば，逆リソース表更新のためのコストは，より小さくなる．

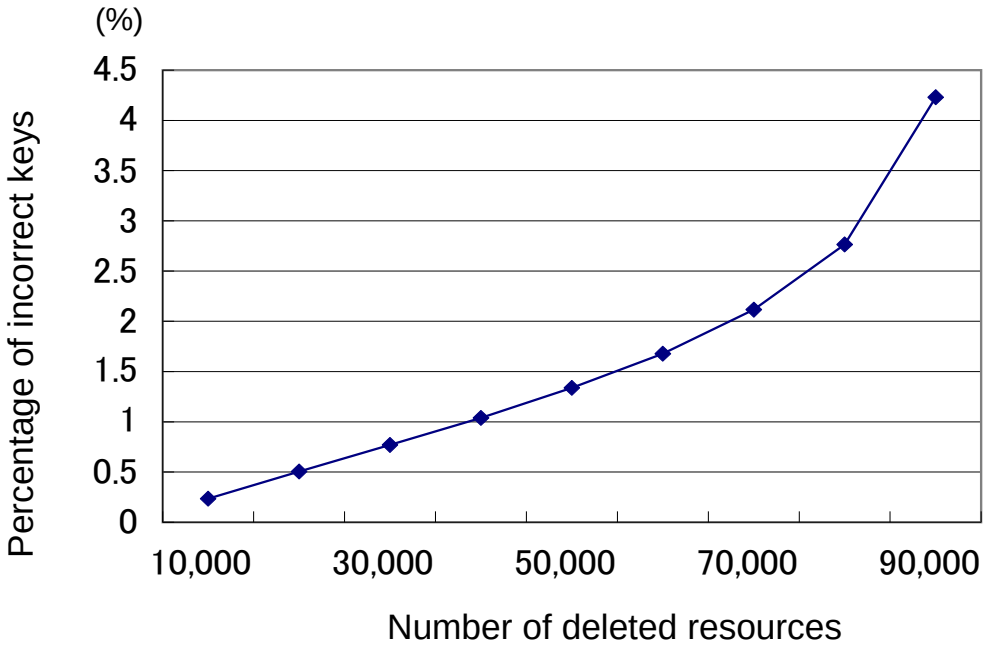


図 15　削除されたリソース数と正しくないキーが占める割合の関係

3.4.2 記憶容量についての評価

順リソース表を記憶しておくのに必要な記憶容量は，通常の DHT と同様であり，全てのリソースの属性情報量とほぼ等しい．一方，逆リソース表では，リソース ID を複数のキーに重複して保存するため，通常の DHT よりも多くの記憶容量を必要とする．そこで，逆リソース表に要する記憶容量を調べるため，前記と同様のシミュレーションを行った．

T=1000，M=20 とし，リソース数 N を様々に変化させた場合の，1 リソースあたりの重複登録数を図 16 に示す．パラメータの変化のさせ方は 3.4.1.2 の場合と同様である．重複登録数は，あるリソースが何箇所のタプルに重複して登録されたかを示しており，リソース 1 つあたりの準候補キー数とも言える．こちらも 3.4.1.2 と同様の曲線を描いており，記憶容量の面でもスケール性があると言えよう．

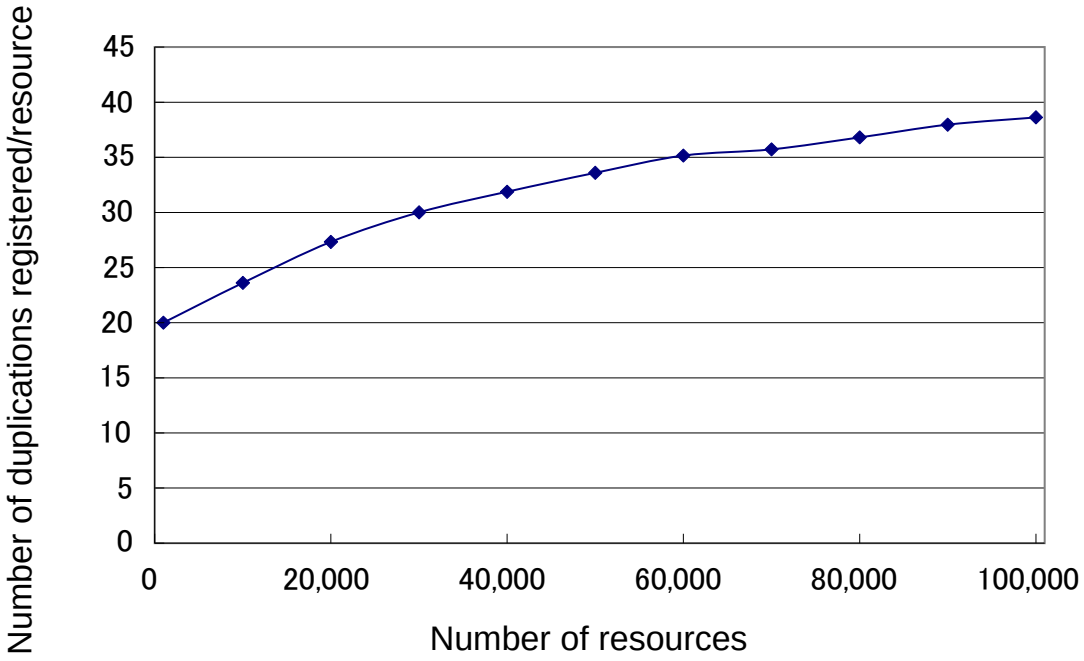


図 16　リソース数 N と重複登録数の関係

3.4.3 閾値 T の設定方法について

閾値 T の値はシステム設計者が設定することを想定しているため，設計する際の指針について明らかにする必要がある．

N=10000，M=20 とし，閾値 T を様々に変化させた場合の，1 リソースあたりの登録試行数，重複登録数をそれぞれ図 17，図 18 に示す．これらより，閾値 T が小さい場合には，T を超えるキーが多くなり，それらの組合せの数だけ新たなキーへ登録されるため，登録試行数，重複登録数共に多くなることが分かる．

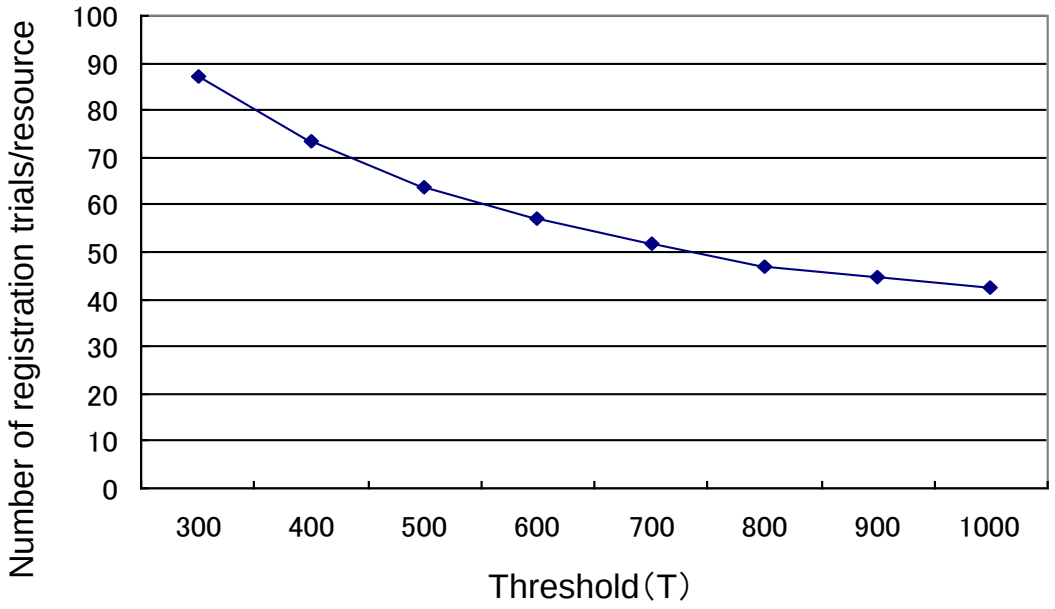


図 17　閾値 T と登録試行数の関係

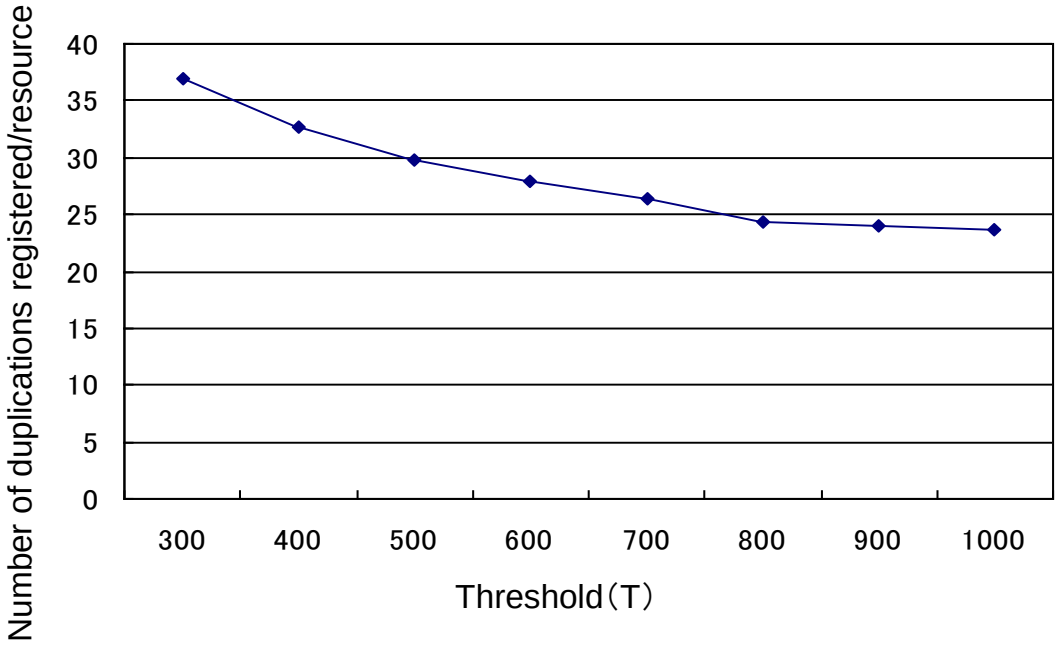


図 18　閾値 T と重複登録数の関係

図 17，図 18，そしてこれまでの評価結果を踏まえると，設定する際には以下の点に注意を払う必要があると言える．

T が大き過ぎる場合

- 検索処理における，中間解のための通信量が大きくなる
- 検索処理における検査処理に要する計算量が大きくなる

T が小さ過ぎる場合

- 溢れが多く発生し，登録処理に要する通信量が大きくなる
- 溢れが多く発生し，逆リソース表のための記憶容量が大きくなる
- ユーザが強い条件を出すことを強いられる

設計者は，利用可能な通信帯域や記憶容量はもちろんのこと，ユーザのニーズ（検索頻度，一度に検索したいリソース数），リソース数やそれらに与えられた属性情報の性質を把握した上で，上記の点に注意して閾値 T の値を決定する必要がある．

3.5 関連研究

DHT におけるキーワード検索に要する通信量を小さくするため，既にいくつかの研究が行われている．

Reynolds らは中間解を Bloom Filter 化し，これを送ることで通信量の圧縮を行うことを提案している [35]．Li らは，Reynolds らの技術も含め，既存技術を調べた結果，クラスタリング，Gap Compression [36]，Adaptive Set Intersection [37]の 3 つを組み合わせる手法が最も検索の通信量を削減できると述べている [38]．しかしながら，それらを組み合わせた場合でも，なお十分に通信量は小さくはならず，Web 検索エンジンを DHT ベースの P2P 検索システムで実現することは困難であるとも述べている．提案方式は，検索結果が T 個以下に絞られる検索条件にしか対応できないなどの制約があるため，Web 検索エンジンを実現するものではないが，中間解を不要とするため，T を低く設定することでこれらの手法より検索の通信量をより低く抑えることが可能であると考えられる．

他にも，本稿における提案方式と同様に，「X 社」と「ネットワークカメラ」などの論理積を予め計算し，1 つの検索キーとしてインデックスに登録しておく（これを Precomputation と呼ぶ）ことにより，通信量を抑えようという方式も提案されている [38]

[39]．このアプローチで問題となるのは，キーワードの組み合わせをもインデックスとするために，インデックスの量が爆発してしまうことにある．

このため，[39]では，2 つのキーワードの組合せのみを Precomputation しておく手法を提案している．3 つ以上の組合せはインデックスとしないことで，爆発を防ぐものである．[39]の方式に比べ，提案方式には以下のような優位点がある．

無駄なインデックスを生成しない：例えば $A_1 \wedge A_2$ という検索条件に対し，A_1 もしくは A_2 が準候補キーである場合，提案方式ではその準候補キーに結び付けられたリソース群を取り出し，その検査処理を行わなければならないが，[39]の方式であれば，$A_1 \wedge A_2$ というキーに結びつけられたリソース群を取り出すだけで良い．しかしながら，こうした検索条件は元々大きな通信を必要とせず，高速処理が可能なクエリである．そうした検索条件のためにインデックスを作成することは，登録処理量の面からも記憶容量の面からも，効率的でない．対して提案方式は，そうした検索条件に対してインデックスを生成しない．

どのような検索条件に対しても，通信量・処理時間を一定量以下に抑える：[39]の方式では 3 つ以上のキーワードを含む AND 条件に対しては，依然として大きな通信が発生してしまう場合がある．対して提案方式は，検索結果が T 個以下に絞られる検索条件でさえあれば，どのような検索条件に対しても，一定量以下の通信量・処理時間に抑えることができる．

[39]の方式が優位な点としては，登録処理の通信量が挙げられる．あるリソースが 20 個の属性情報を持っているとすると，[39]の方式では，$20+{}_{20}C_2$，すなわち 210 回の登録試行を必要とし，これはリソース数に依存しない．図 14 と比較し，その回帰式を元に計算すると，リソース数が約 3 千万個以上になると提案方式の方が多くの登録試行数を要することが分かる．とは言え，提案方式においても，登録試行数はリソース数の増加に対して *log* オーダで伸びるだけであり，スケール性があることに変わりはない．

一方 [38]では，検索条件として頻繁に使われるキーワード同士のみ，組み合わせてインデックスとする手法を提案している．しかしながらこの手法でも，Precomputation されていない AND 条件に対しては，やはり多くの通信や計算を要し，検索速度の高速性を保つことができない．また，検索条件となるキーワードの使用頻度に大きな偏りがある場合には効果的に通信量を削減できると考えられるが，本稿で想定しているような実世界の事物をリソースとする場合には，各人が検索したいリソースは異なることが多く，使用されるキーワードは，Web 検索の場合などに比べて一層分散すると予想される．そのような場合，こうした使用頻度に基づく Precomputation の効果は薄れてしまう．他にも，中間解や検索結果をキャッシュしておくことにより，通信量を減らそうとするものもある [35] [17]が，同様

の理由により効果的には働かないと考えられる．提案方式では，検索条件となるキーワードの使用頻度の分布に関わらず，通信量を抑えることができる．

また [40]では，各担当ピアにリソースの属性情報全体の複製を保持させ，AND 条件による検索においても，それに含まれる 1 つの属性情報だけで検索し，その担当ピアにおいて検査処理を行うことにより，中間解の通信を不要とする．しかしながら，検査処理が大量になり，計算量や遅延が大きくなってしまう場合があると考えられる．提案方式では，検査処理量も低く抑えることが可能である．

3.6 多次元完全一致検索のためのインデキシング技術まとめ

第 3 章では，M2M データの多次元完全一致検索のための新しいインデキシング技術「準候補キーインデキシング」について提案し，提案手法を用いた DHT ベース P2P 検索システムにより，期待通り検索時の通信量が抑制され，リソース情報（M2M データ）の量に依存せず高速な検索処理を維持できることを示した．また準候補キーインデキシングは，登録時（インデックス構築時）に従来より多くの通信を要し，またインデックス保持には多くの記憶容量を要するが，シミュレーションにより，スケール性の観点で問題がないことを定量的に確認した．リソース情報（M2M データ）の変化に応じたインデックス更新手法についても述べた．また評価結果を元に，システム設計において，重要なパラメータ T を定める際に注意すべきことを述べた．

提案する準候補キーインデキシング技術により，高速性とスケール性，負荷追従性をもつ，M2M データに対する多次元完全一致検索機能の実現が可能となる．これにより，多くの M2M アプリケーションにより必要とされる，実世界上の事物（リソース）を自在に探し出し，利用するための，複数の属性情報をキーワードとするキーワード検索が効率的に実現できると言える．

第 4 章
多次元範囲検索のためのインデキシング技術

4.1 M2M データに対する多次元範囲検索の特徴と提案するインデキシング技術概要

第 3 章では M2M データの多次元完全一致検索のためのインデキシング技術「準候補キーインデキシング」について述べた．これにより，水平統合型プラットフォームにおけるリソースのキーワード検索が可能となる．しかしながら，M2M データは位置や時間，測定値など数値データを含むため，多くの M2M アプリケーションは範囲検索機能を必要とする．特に，多次元範囲検索ができれば，「東経○～○度，北緯○～○度の今日の降雨量データ」のような検索が可能となる．各範囲にラベル付けすることにより，多次元完全一致検索でも類似の範囲検索が可能であるが，当該ラベルに依存した範囲検索しかできず，柔軟性に欠けてしまう．

また水平統合型プラットフォームに蓄積されたデータの再利用性を高めるため，当該多次元範囲検索機能は，多様なデータの中からアプリケーションに有用なデータ全てを効率的に検索可能であるべきである．蓄積されたデータは，デバイスの種類，製造元，バージョンなどにより多様なスキーマを持ち得る．降雨量を示すセンサデータと言っても，あるデータは降雨量，計測時間，計測位置の次元だけを含むが，別のデータはさらに温度や湿度の次元を含む．このとき，例えば地図上に降雨量分布を表示するアプリケーションにとって，時間，位置（緯度 / 経度），降雨量の 4 次元の範囲条件に合致するデータはスキーマによらず有用であり，効率的に検索可能であるべきである（値範囲の指定がなく，ただ「ある次元を含む」という条件も，一般に「その次元について－∞から∞の値をもつ」という範囲条件と考えることができる）．多様なデータの中から，必要とするデータをスキーマ横断的に検索するこのような多次元範囲検索をスキーマレス検索と呼ぶこととする．

スキーマレス検索を効率化する手法は確立されていない．多様なデータを扱うデータベ

ースとして，半構造データを蓄積する XML データベースや，いわゆるスキーマレスデータベース [41] [42]も存在するが，これらは全文検索や 1 次元検索のためのインデックスはサポートするが，スキーマレス検索を効率的に実行することはできない．また一般に，検索条件が事前に分かっている場合には，データベース設計の段階において検索条件に適切なインデックスを設計・構築する．複数の検索条件が行われる場合には，各検索条件毎にインデックスを設計・構築する．しかしながら，想定する水平統合型プラットフォーム環境においては，次々に新しいアプリケーション，新しい検索条件が発生しうるため，事前にインデックスを設計するのは困難である．さらに，各検索条件のために多数のインデックスを構築すると，インデックスの容量や，挿入プロセスにおけるインデックス構築のための計算コストが大きくなってしまう．なるべく多くの検索条件に効果的な，少ない数のインデックスを構築することが理想的である．

そこで本章では，高速性とスケール性，負荷追従性をもち，M2M データに対する多次元範囲検索，特にスキーマレス検索を可能とする新しいインデキシング技術「UBI-Tree」を提案する．異なるスキーマをもつデータであっても 1 つの木構造でインデキシングすることができ，また多様な多次元範囲検索を効率化するインデキシング技術である．UBI-Tree は R-Tree [43]をベースとした木構造によりデータをインデキシングする．UBI-Tree の挿入アルゴリズムにおいては，新たに導入した 2 種の「分類の悪さを示す指標」を用いて，2 段階で多様なデータを各ノードに分類する．これらの指標は，各データが含んでいる次元の種類の違い自体を表現するものであり，次元の数や種類が異なるデータが混在した場合にも定義可能である．それゆえ，UBI-Tree は異なるスキーマをもつデータが混在するデータもインデキシングすることができる．さらに，似た次元を持つデータを同じノードに分類することにより，UBI-Tree は次元の呪いの発生を抑え，また似た値を持つデータを同じノードに分類し，スキーマレス検索を効率化する．

図 6 に，多次元範囲検索技術における UBI-Tree 技術の位置付けを示す．図 6 に示すように，現在最も利用されているリレーショナルデータベース（Relational Database; RDB）は，スキーマが統一された帳票データ等を主な対象としている．RDB においては，範囲検索の効率化には多くの場合 B-Tree（あるいはその派生技術）が用いられ，また多次元範囲検索には R-Tree（あるいはその派生技術）等が用いられる場合もある．一方，XML-DB の他，近年スキーマレスな（不定形な）データを扱うデータベースとして CouchDB や MongoDB が注目を集めている．これらのデータベースでは，1 次元範囲検索の効率化に B-Tree が用いられている．M2M データはスキーマレスなデータであり，また代表的な M2M データであるセンサデータ等は多次元連続値をもつため，スキーマレスデータに対する多次元範囲検索が必要となる．UBI-Tree は，これまでのインデキシング技術では対応できなかった，当該検索の効率化を実現する技術である．なお，多次元範囲検索のための従来のインデキシング技術は，スキーマフルなデータに対するものであり，画像・映像の特徴量による検索を可能とするマルチメディアデータベース等において用い

られる．

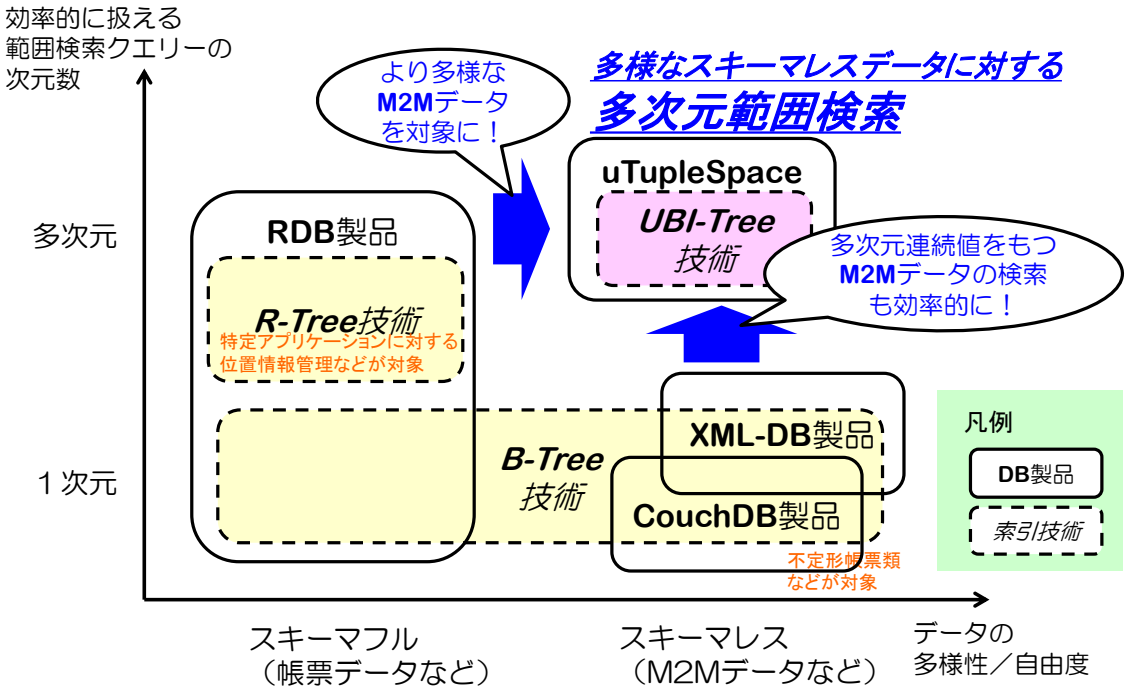


図 6（再掲）多次元範囲検索技術における UBI-Tree 技術の位置づけ

本章の構成は以下の通りである．4.2 では，UBI-Tree の基本的アルゴリズムを提案する．また 4.3 では，追加アルゴリズムにより負荷追従性を持たせた UBI-Tree（Load-Adaptive UBI-Tree）について提案する．4.4 では更に，UBI-Tree のスケール化のための新しい負荷分散方式について提案する．各節においては，方式やアルゴリズムの詳細について述べるとともに，既存技術との比較や，評価結果について述べる．

4.2 多次元範囲検索のための UBI-Tree インデキシング技術の検討

4.2.1 従来技術とその問題

B-Tree，あるいはその派生技術は，現在，最も用いられている木構造インデックスの 1 つ

である．B-Tree は 1 次元インデックスであるが，しばしば多次元検索にも用いられる．B-Tree を用いて多次元検索を実行する 1 つの手法は以下の通りである．事前に，検索条件に用いられる次元の 1 つを B-Tree によりインデキシングしておく．検索時には，まず B-Tree を用いてその 1 次元によりデータを絞り込んで中間解を取得し，さらに中間解から他の次元について条件に合致しないデータをフィルタリングすることで最終的な検索結果を得る．B-Tree を用いて多次元検索を実行するもう 1 つの手法は，事前に全次元を B-Tree によりインデキシングしておき，検索時に各 B-Tree の結果をマージすることにより検索結果を得るというものである．しかしながら，各 B-Tree の結果が十分に絞り込めない場合，フィルタリングやマージによる処理負荷はそれだけ大きくなってしまう.また半構造データの値や構造に対するインデキシング技術 [44] [45]も提案されているが，複数インデックスを用いるためにやはりフィルタリングやマージによる処理負荷が大きい，あるいは転置リストを用いたインデキシングであるため範囲検索ができない，などの問題がある．

一方，多次元検索のためのインデキシング技術も数多く存在する [43] [46] [47] [48] [49]．しかしながら，高次元データをインデキシングした場合には，「次元の呪い」が問題となる．各センサデータは温度や照度といった計測値とともに，計測時間，計測位置，デバイス ID などを含む．各データはせいぜい 10 数次元を含むだけであるが，その多様性によりデータ全体での次元数は増加し，例えば数百次元に及ぶため，次元の呪いが発生する．これにより検索効率は低下し，それらの手法を用いても線形探索と同等の効率となってしまう．データが高次元になるとデータ間の距離の比が漸近的に 1 に近づき，データのクラスタリングが本質的に困難になることが原因であり，このような現象を次元の呪いと呼ぶ．近年，近似近傍検索の効率化において，次元の呪いを解決する手法が提案された [50] [51]．高次元空間内の検索点に対する近傍点を検索する近傍検索は，検索条件に全次元を同じ値範囲で指定する多次元範囲検索と似た検索タスクであり，当該多次元範囲検索の効率化にこれら従来手法を応用することが考えられる．しかしながら，スキーマレス検索では，検索条件によって指定される次元の種類は異なり，また次元によって指定される値範囲は異なるため，近傍検索の手法を多様なスキーマレス検索の効率化に用いることは困難である．

またそもそも，多次元範囲検索のための従来手法 [43] [46] [49]は，異種異数次元を含むデータや検索条件をうまく扱うことができない．それらは同種同数次元を含むデータを前提としているため，異種異数次元を含むデータにおいてはデータ間の距離を決定することができない．たとえ各データが全次元を保持するよう，なんらかの値で存在しない次元を補完してインデキシングしたとしても，多くの場合データを適切に分類することはできない．また同種同数次元を含む検索条件を前提としているため，各次元が検索条件に用いられる頻度の差異をうまく考慮することができない．例えば多くの M2M アプリケーションにおいて，時間や位置といった一般的な次元は検索条件として多く用いられる一方，降雨量などセンサ種別に固有な次元は特定の M2M アプリケーションでのみ用いられる次元となると考えられる．このような場合，検索条件によく用いられる次元を重視して分類することで全

体的な検索効率の向上につながると考えられる．

他にも，多次元検索のための，grid-file [52]などの空間分割手法もある．しかしながら，センサデータは全体では高次元であるものの，各センサデータはせいぜい数次元であるため，無駄なグリッドが数多く発生してしまい，検索効率は劣化する．さらに，次元種類数の増加に伴う空間拡張処理の処理コストが高いと考えられる．また VA-file [53]は各データを圧縮することにより，高次元線形探索を効率化する．しかしながら，センサデータのようなデータは元々サイズが小さいため，その改善効果は小さいと考えられる．Local Dimensionality Reduction（LDR） [54]と呼ばれる手法は，局所相関に基づき高次元データをクラスタリングし，各クラスタにおいて次元縮退を行うことで次元の呪いを抑制する．しかしながら，本手法も同種同数次元の高次元データと近傍検索を前提としており，またクラスタリングの計算量が大きいと思われる．

以上より，効率的なスキーマレス検索を実現するためには，次元の呪いの発生を抑え，また異種異数次元を含むデータや検索条件を効率的に扱うことができる新しい手法が必要であると考えられる．

4.2.2 提案手法「UBI-Tree」

4.2.2.1 基本的アイデア

多次元インデキシング技術として最もシンプルかつ代表的な R-Tree をベースとする新しいインデキシング技術 UBI-Tree を提案する．UBI-Tree は，「検索時にアクセスされる確率」と「次元種類数」の 2 つを，データ分類の悪さを示す指標（コスト）として用いることでスキーマレス検索を効率化する．

各ノードの「検索時にアクセスされる確率」をコストとし，これが増加しないようにデータを分類する．これにより，検索時のアクセスノード数が減少し，効率的な検索が可能となる．木構造インデックスにおいて検索効率を向上させるためには，検索時にうまく枝刈りすること，つまりいかに少ないノードだけをアクセスするようにできるかが重要である．そのためには，各ノードの「検索時にアクセスされる確率」を正しく見積もり，これを小さくすることが重要である．そこでコスト算出においては，各次元が検索条件に用いられる頻度の差異も考慮する．これにより，検索条件によく用いられる次元を重視し，その次元の値が近いデータ同士ほど同じノードに分類されるようになる．

また各ノードの「次元種類数」をコストとし，これが増加しないようにデータを分類することで，同じ次元をもつデータが同じノードに分類される．これにより，リーフノードに近づくにつれ各ノードの次元種類数を急激に減少させることができ，次元の呪いを抑えることができる．個々の M2M データは元々低次元であるから，この特性を活かし，次元種類数

の計数という単純な処理のみで効率的に次元縮退を行うことが可能となる．また「次元種類数」による分類は「検索時にアクセスされる確率」による分類に先んじて行う．これにより，「次元種類数」による分類により各データ種別のデータはほぼ別々の部分木に分類され，各部分木内で「検索時にアクセスされる確率」による分類が行われるようになる．各データ種別のデータ量に偏りがある場合，データ量の大きいデータ種別は大きな部分木に分類され，少ないデータ量のデータ種別は小さな部分木に分類される．大きな部分木に分類されたデータは，同じデータ種別内でも特に値が近いデータ同士が同じノードに分類されるよう「検索時にアクセスされる確率」により部分木内でさらによく分類されることになる．各データ種別のデータ量には多くの場合偏りがあると想定されるが，次元種類による分類と値による分類をシームレスに統合する本手法により，そのようなデータをうまく収容・分類することが可能となる．

UBI-Tree は R-Tree と同様に，データを最小包囲矩形（MBR; minimum bounding rectangle）と呼ばれる方形領域に分割するバランス木である．R-Tree などの従来のインデキシング技術は，各ノードの MBR に対応する超直方体体積や超直方体間の重なり部分の体積等をコストとし，そのコストが増加しないように検索木を構築する．しかしながら，UBI-Tree は多様な種類や数の次元を含むデータをインデキシングするため，MBR の次元もノードによって様々となる．それゆえ，それらの超直方体体積等を比較することは無意味である．これに対し UBI-Tree で用いる 2 つのコストは，異種異数次元を含むデータ集合間でも定義することができる．

UBI-Tree は R-Tree と同様に，木を構成する各ノードは 1 つ以上のエントリを含む．リーフノードにおいてはエントリはデータである．インナーノードにおいてはエントリは子ノードへのポインタとその子ノードの MBR 情報のセットである．MBR 情報は，そのノード，あるいはそのノードより下位にあるノードが保持するデータに含まれる各次元の値範囲の情報を含む．R-Tree と UBI-Tree の違いを図 19 に示す．UBI-Tree は多様なデータをインデキシングするため，UBI-Tree のエントリは多様な次元をもつ．挿入時には，検索時と同様にルートノードから枝を辿っていく点は R-Tree と同様であるが，辿る枝を選ぶために前述の指標を用いる点で R-tree と異なる．

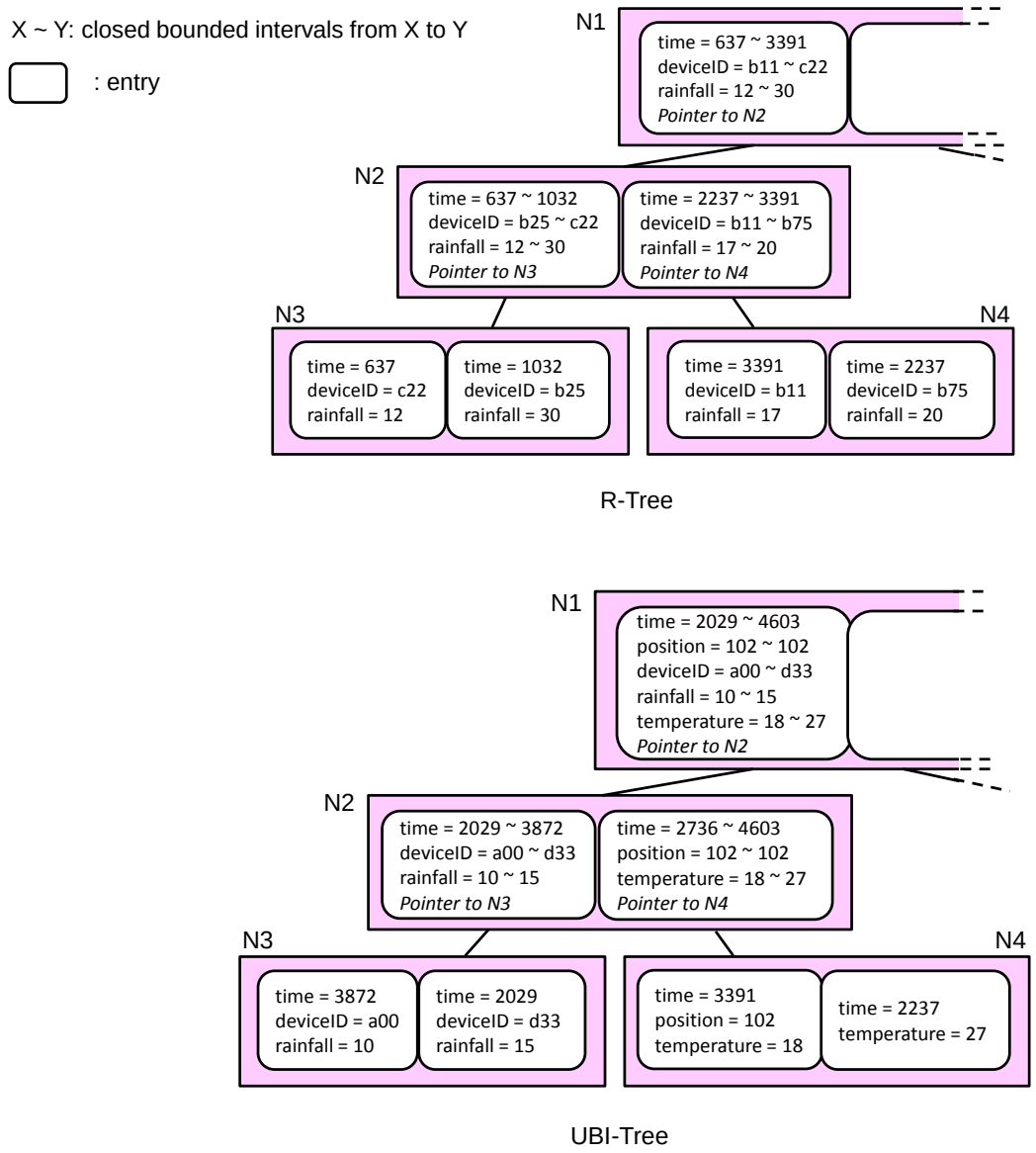


図 19　R-Tree と UBI-Tree の違い（木の右側は省略）

以下，UBI-Tree における挿入プロセス，検索プロセス，またデータ構造について詳細に述べていく．

4.2.2.2 挿入プロセス

一般に，木構造をもつインデックスにおける挿入プロセス（Insert）は，ルートノードから開始し，挿入先子ノード選択アルゴリズム（ChooseSubtree）を用いて選択した子ノード

を再帰的に辿り，最終的に辿り付いたリーフノードへデータを挿入する [55]．各ノードが含むエントリ数には制約が設けられており，最小値は E_{min}，最大値は E_{max} である．E_{max} を越えた場合には，そのノードを 2 つのノードに分割し，エントリは各ノードに分配する．ノードの分割はノード分割アルゴリズム（Split）に従って実行される．

UBI-Tree の挿入プロセスは，挿入先子ノード選択アルゴリズムとノード分割アルゴリズムにおいて，従来と異なるコストを用いる点に特徴がある．それぞれ概要を以下に示す．

挿入先子ノード選択アルゴリズム概要（ChooseSubtree）：

CS1　「次元種類数」の増加が最小となる子ノードを選択する

CS2　CS1 で該当する子ノードが複数存在する場合には，さらにその中から「検索時にアクセスされる確率」の増加が最小となる子ノードを選択する

ノード分割アルゴリズム概要（Split）：

SP1　分割後の 2 つのノードの「次元種類数」の和がなるべく小さくなるように分割する

SP2　SP1 で該当する分割方法が複数存在する場合には，さらにその中から分割後の 2 つのノードの「検索時にアクセスされる確率」の和がなるべく小さくなる分割方法を選び，分割する

以下，やや複雑な「検索時にアクセスされる確率」の算出方法について説明し，さらに，挿入プロセスを高速化するための手法について説明する．

4.2.2.2.1 「検索時にアクセスされる確率」の算出方法

まず，検索条件に次元集合 S が用いられる確率 $P_c(S)$を次式で求める．

$$P_c(S) = \frac{1}{M} \sum_{t_s \in T_s} \frac{1}{2^{V_{t_s}} - 1} \tag{2}$$

ただし，T_s は次元集合 S を含む蓄積データの部分集合を表し，t_s は T_s の要素である．またV_{t_s} は t_s に含まれる次元種類数であり，M は蓄積データ数を表わす．例えば，5 件のデータ[x=1]，[x=2]，[x=3, y=4]，[y=5]，[x=6, y=7]が蓄積されている場合，各次元が検索条件に用いられる確率は式 (2) を用いて以下のように算出される．

$$P_c(x) = \frac{1 + 1 + 1/3 + 1/3}{5} = \frac{8}{15}$$

$$P_c(y) = \frac{1/3 + 1 + 1/3}{5} = \frac{5}{15} \tag{3}$$

$$P_c(x, y) = \frac{1/3 + 1/3}{5} = \frac{2}{15}$$

式 (2) は以下の考え方に基づいている．近傍検索とは異なり，スキーマレス検索は全次元を常に検索条件に用いるとは限らない．つまり，各次元が検索条件に用いられる確率は等しくない．新しいデバイス，新しいアプリケーションが動的に出現する M2M 環境においては，どのような検索条件がどのような確率で用いられるかを予測することは困難である．しかしながら，ある次元は 1 万件のデータに含まれ，また別の次元は数件のデータにしか含まれないのであれば，前者の次元は後者の次元に比べ，検索条件に用いられる可能性が高いと予測される．例えば，時間や位置といった次元は多くのセンサデータに含まれ，よく検索に用いられるが，降雨量のような次元を含むセンサデータは少なく，検索に用いられることも少ない．そこで，以下の仮定を置くことにより，式 (2) が導かれる．

- 蓄積された各データが検索の目的データとなる確率は等しい（上記例で言えば，1〜5 件目のデータそれぞれを検索により取得したいという検索要求は等確率で発生する）．
- あるデータを検索したい場合，当該データが含む次元集合の，ある部分集合を用いて検索するが，このとき各部分集合が検索に用いられる確率は全て等しい（ただし空集合を除く．例えばデータ[x=3, y=4]を検索したい場合，部分集合(x)，(y)，(x, y)が検索に用いられる確率はそれぞれ 1/3 となる）．
- ある次元集合が検索に用いられる確率は，その次元集合を含む各蓄積データが，検索の目的データとなり，かつ当該次元集合が検索に用いられる確率の単純な総和に等しい．

次に，検索条件に次元 D が用いられた場合に，ノード N がその検索によりアクセスされる確率 $P_N(D)$を次式で見積もる．

$$P_N(D) = \frac{2(\beta - D_{min})(D_{max} - \alpha) - (\beta - \alpha)^2}{(D_{max} - D_{min})^2} \tag{4}$$

ただし，D_{min}～D_{max} は D の定義域を表し，α～βはノード N の MBR が D において占める領域を表す．ここで x～y は x から y までの閉区間を表わしている．

式 (4) は以下の考え方に基づいている．D の値が D_{start} 以上 D_{end} 以下であるという検索条件を「$D = Dstart$～D_{end}」で表す．このとき，(D_{start}, D_{end})を 2 次元座標空間上の点と考えると，D に関するすべての検索条件は図 20 に示した上三角形領域内の点としてプロットすることができる [56]．これらの点が上三角形領域内に限定されるのは，以下 3 つの式が一般に成り立つからである．

$$D_{start} \leq D_{end}$$
$$D_{start} \geq D_{min} \qquad (5)$$
$$D_{end} \leq D_{max}$$

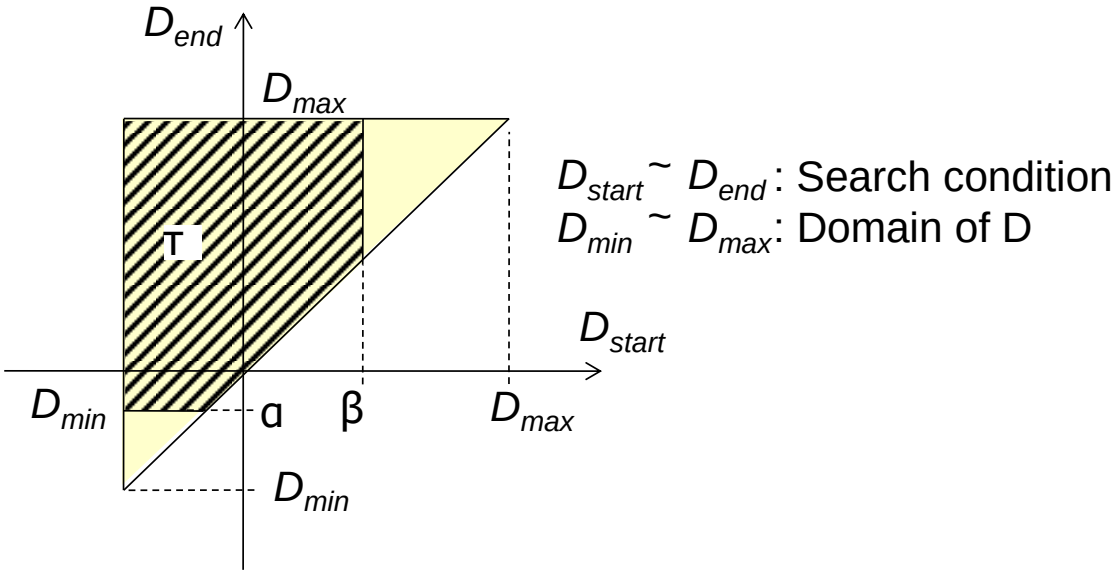


図 20　次元 D に関する範囲検索条件

ノード N の MBR が D についてα～βを占める場合，図 20 に示した斜線領域 τ 内にプロットされる検索条件は，ノード N にアクセスする必要がある．検索範囲と MBR に重複部分が存在し，検索条件に合致するデータがノード N，あるいはその子孫ノードに存在する可能性があるためである．つまり D が検索条件に用いられた場合，その検索条件が領域 τ 内にある確率が，その検索においてノード N がアクセスされる確率となる．上三角形領域内の各点に対応する検索条件が等確率で発生すると仮定すると，この検索においてノード N

がアクセスされる確率を「τ の面積 / 上三角形全体の面積」で見積もることが可能となる．

なお式 (4) において，D の定義域 D_{min}〜D_{max} が不明である場合，あるいは定義域の下限・上限が無限小・無限大の場合には，D_{min}，D_{max} として蓄積データ全体における次元 D の最小値，最大値を用いれば良い．ただしこの場合，「定義域外の値を含む範囲の検索条件は発生しない」ではなく「蓄積データ全体における次元 D の最小値よりも小さな値，あるいは最大値よりも大きな値を含む範囲の検索条件は発生しない」と仮定し，$P_N(D)$を求めることとなる．

式 (2) と式 (4) により，検索時にノード N がアクセスされる確率 P は次式で表現できる．

$$P = \sum_{S \subseteq U} P_c(S) \prod_{D \in S} P_N(D) \tag{6}$$

ただし，U は蓄積データ内に存在する次元の全組合せを表す．

4.2.2.2.2 挿入プロセスの高速化手法

挿入プロセスを高速化して実用に耐えうるものとするため，以下に述べる 2 つの改善を行う．

1 つ目の改善は，式 (6) の計算の速度向上である．まず，式 (6) に含まれる式 (2) ($P_c(S)$) において，S の種類数は蓄積データに出現する次元の組合せの数だけ存在するため，全ての S について式 (2) を計算するのは計算量が大き過ぎる．そこで，1 つの次元のみを含む S について式 (2) を計算し，それらから複数次元を含む S の要素に対する式 (2) を近似的に算出することとする．例えば $P_c(x, y)$は $P_c(x) \cdot P_c(y)$で近似する．次に，本近似値を用いて式 (6) を計算すると，複数次元を含む S に対応する項の値は小さくなるため，これを無視しても大きく特性は変わらないと考えられる．そこで，複数次元を含む S に対応する項を 0 で近似する．これらの近似を併せると，P の近似値 P'を式 (7) で見積もることとなる．ただし，S_{all} は蓄積データ内に含まれる全種類の次元を含む次元集合である．本近似により，挿入先子ノード選択結果等への影響をなるべく抑えつつ，計算量を次元種類数に比例する量に抑えることが可能となる．

$$P' = \sum_{D \in S_{all}} P_c(D) P_N(D) \tag{7}$$

2 つ目の改善は, ノード分割アルゴリズムの速度向上である. ノード分割アルゴリズムは, より良い検索効率のために最も良いノード分割方法を選択しようとするものであるが, ノード分割方法はエントリ数の組合せの数$2^{E_{max}}$だけ存在する. 4.2.3.2.2 で述べるように良い検索効率を実現するためには E_{max} は 20 程度とする必要があるため, 総当たりで調べる方法では非常に計算コストが高い. 最適に近い分割方法を近似的に見つけ出す方法が求められる. そこで, R-Tree [43]で提案された Quadratic-Cost Algorithm を UBI-Tree 向けに修正したアルゴリズムを導入する. 本アルゴリズムは, 分割後の 2 つのノードの次元種類数や検索時にアクセスされる確率がなるべく小さくなるようノードを分割する. まず, 分割すべきノードのエントリ群の中から, 同じノードに入れた場合に最も次元種類数や検索時にアクセスされる確率が増加してしまう 2 つのエントリを選出する. その後, 選出した 2 つのエントリをそれぞれ分割後の新たなノード 2 つに 1 つずつ挿入し, さらに残りのエントリを 1 つずつどちらかのノードに振り分ける. このとき, どちらのノードに振り分けるかにより, コストの増加に最も大きな差異がある順に振り分ける. 以下, アルゴリズム詳細を示す. ただし, *E*(*node*)はノード *node* のエントリ数を示す.

ノード分割アルゴリズム詳細 (Split Detail):

SPD1　新しい 2 つのノードを作成し, それぞれ *N1*, *N2* と呼ぶ. また分割すべきノードを *N* と呼ぶ.

SPD2　初期エントリ選択アルゴリズム（PickSeeds）を用いて選択した 2 つのエントリを *N* から取り出し, それぞれ *N1*, *N2* に挿入する.

SPD3　$(E_{min} - E(N1)) \geq E(N)$ となる場合は, *N* 内の残りのエントリを全て取り出し, *N1* に挿入する.

SPD4　$(E_{min} - E(N2)) \geq E(N)$ となる場合は, *N* 内の残りのエントリを全て取り出し, *N2* に挿入する.

SPD5　*N* 内にエントリが残っている場合, 次回エントリ選択アルゴリズム（PickNext）を用いて選択したエントリを *N* から取り出し, 同じく次回エントリ選択アルゴリズムにより選択した挿入先の *N1* または *N2* に挿入する.

SPD6　*N* 内にエントリが残っている場合, SPD3 に戻る.

SPD7　*N* の親ノードから *N* のエントリを削除し, 代わりに *N1* と *N2* のエントリを挿入する.

初期エントリ選択アルゴリズム (PickSeeds):

PS1　N 内のエントリの全ペア(2 つのエントリを選び出す全組み合わせ)について,

「一方のエントリしか保持しない次元の種類数」を算出し，その値が最も大きいペアを全て見つけ出し，リスト L1 に保持する．

PS2　リスト L1 に含まれる全ペアについて，「((当該ペアを挿入したノードが検索時にアクセスされる確率）－（当該ペアの一方のエントリを挿入したノードが検索時にアクセスされる確率）－（当該ペアの他方のエントリを挿入したノードが検索時にアクセスされる確率))」を算出し，その値が最大となるペアを選択する．最大となるペアが複数存在する場合には，そのうち任意の 1 つを選択する．

次回エントリ選択アルゴリズム (PickNext)：

PN1　*N* 内に残っている各エントリについて，「｜（当該エントリを *N1* に挿入することによる次元種類増加数）－（当該エントリを *N2* に挿入することによる次元種類増加数）｜」を算出し，その値が最も大きいエントリを全て見つけ出し，リスト L2 に保持する．このとき，どちらのノードに挿入した方が次元種類増加数が小さいかを示すノード情報 T（*N1* または *N2*）を，当該エントリと合わせてリスト L2 に保持する．

PN2　リスト L2 に含まれる全エントリについて，「((当該エントリをノード T に挿入した場合の，当該ノードが検索時にアクセスされる確率の増加量）－（当該エントリを T でないノード(例えば T=*N1* の場合であれば *N2*)に挿入した場合の，当該ノードが検索時にアクセスされる確率の増加量))」を算出し，その値が最小となるエントリを選択する．最小となるエントリが複数存在する場合には，そのうち任意の 1 つを選択する．またこのとき，挿入先ノードとして T を選択する．

4.2.2.3 検索プロセス

検索アルゴリズム（Search）は一般の木構造インデックスと同じである．検索条件に含まれる全ての条件を満たす子ノードを再帰的に辿り，目的のデータを見つけ出す．この際，調べるべき次元は検索条件に含まれる次元だけでよい．

4.2.2.4 データ構造

インデックスの高速動作を実現するためには，インデックスサイズをなるべく小さく抑え，オンメモリで動作させられるようにすべきである．そこで本節では，UBI-Tree のデータ構造を簡約化する方式について述べる．

一般に R-Tree などの検索木を構成する各ノードは，MBR 情報，すなわち子ノード以下に含まれるデータの各次元の範囲情報を表形式で保持している（範囲表と呼ぶ）．R-Tree の

データ構造を図 21 に示す．データ挿入 / 検索時には，各ノードの範囲表を参照し，前述したアルゴリズムに従って子ノードを選択しながらルートノードからリーフノードへ木を辿っていく．例えば温度が 10 のセンサデータを検索する場合，まず行番号表（Line number table）から温度の値範囲情報が各範囲表の 3 行目に記載されていることを確認し，ルートノードから検索を開始する．このとき，各ノードの範囲表を確認しつつ，求めるデータを含みうる枝だけを辿っていく．例えばノード *N0* の範囲表を確認すると，ノード *N1* の温度の値範囲は「15～23」であるため，ノード *N1* 以下に温度が 10 のデータは存在しないためノード *N1* 以下の検索は取りやめる一方，ノード *N2* の温度の値範囲は「10～17」であるため，求めるデータを含みうると判断し，ノード *N2* 以下については検索を続行する．このように，求めるデータを含まないノードは辿らず（一般に「枝狩りをする」と表現する），求めるデータを含みうるノードだけを検索していくことで，効率的に検索することが可能である．

しかしながら，多様なセンサを扱う場合，扱う次元数が増大するとともに範囲表のサイズも大きくなるため，検索木をオンメモリに保持できなくなり処理速度が極端に低下してしまう．

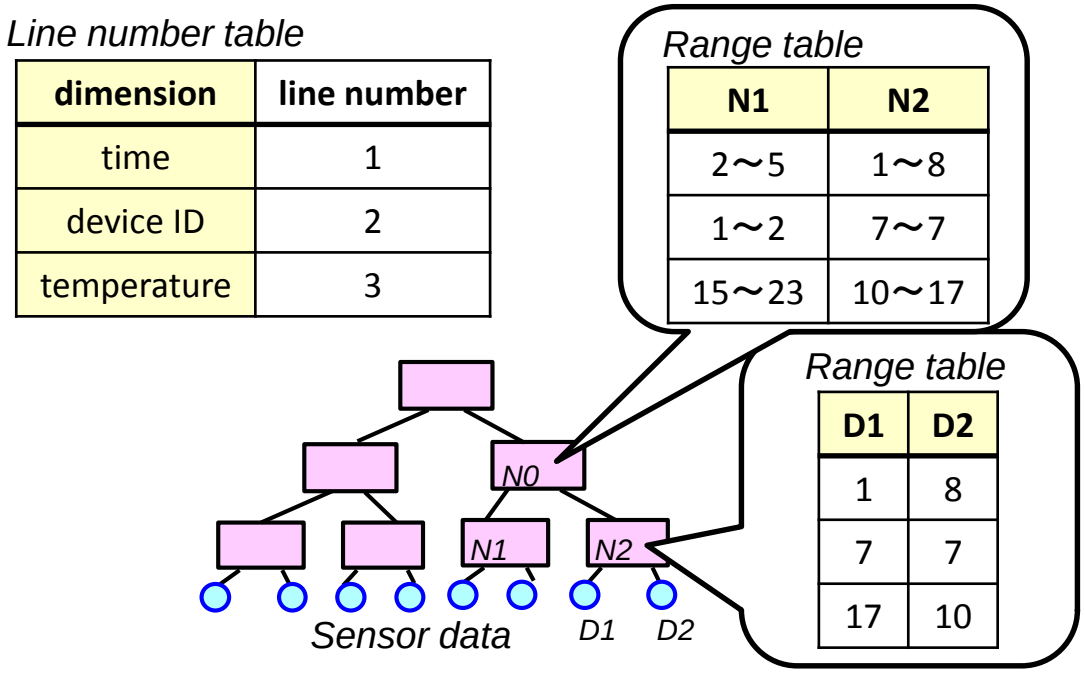


図 21　R-Tree のデータ構造

しかしながら本研究でターゲットとするセンサデータ集合は，その多様性のために全体

では数百次元に及ぶが，個々のデータは数次元に過ぎない．さらに前述したように，UBI-Tree は同じ次元をもつデータを同じ子ノードへ振り分けることにより，検索処理を効率化している．このため，R-Tree と同様のデータ構造で UBI-Tree を構築した場合，図 22 に示すように，リーフノードに近づくにつれ各ノードが保持する次元の種類が特定のものだけに絞り込まれ，範囲表は空欄の多い，スパースなものとなる．例えば図 22 のノード N2 の範囲表を見ると，空欄が多いことが分かる．そこで，UBI-Tree においてはセンサデータが持つこのスパース性を利用し，図 23 に示すように簡約化した範囲表を用いる．範囲表には新たに 1 カラムを加え，子ノードの範囲表において対応する行の番号を格納することで，存在しない次元に該当する行を省略可能とする．データ挿入 / 検索時に検索木を辿る際には，行番号を記載したカラムを参照することで，子ノードにおいてどの次元がどの行に対応するかを判断することができる．例えば図 23 のノード *N0* の範囲表において，5 行目に記載された加速度（acceleration）の情報を見ると，ノード *N2* においては加速度の値範囲が「-0.4～1.5」であるとともに，ノード *N2* の範囲表では加速度の情報が「3」行目に記載されていることが分かる．

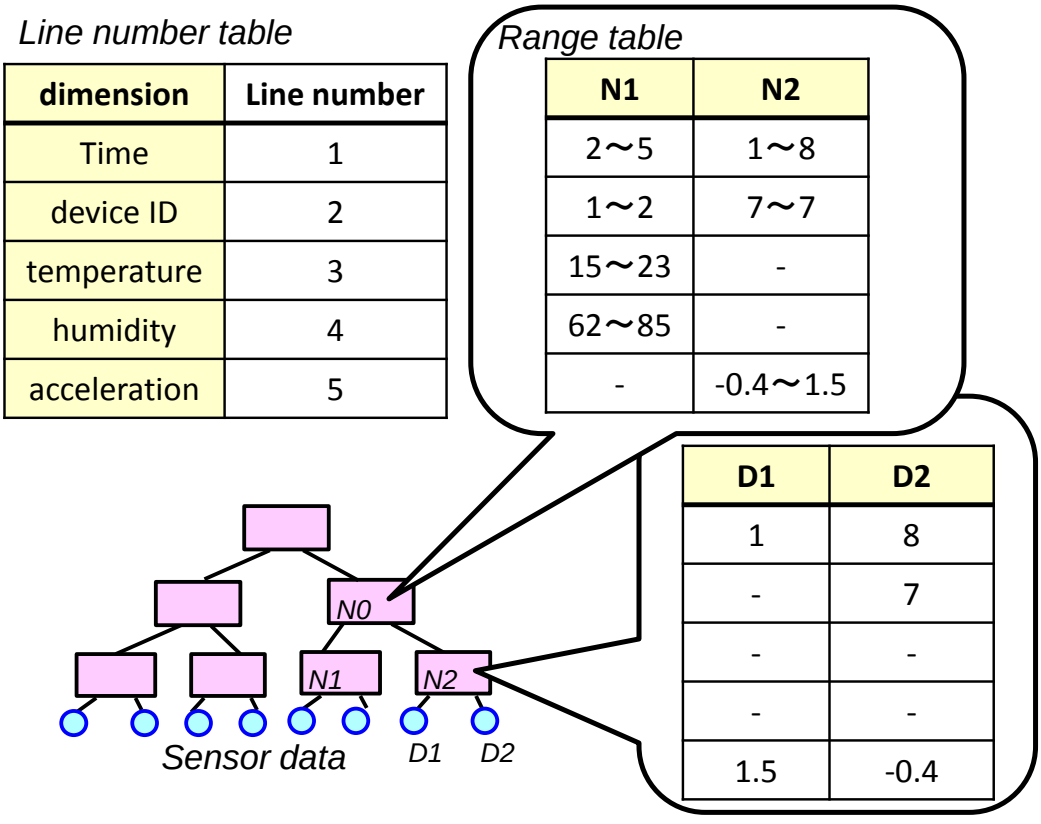


図 22　R-Tree と同様のデータ構造を用いた UBI-Tree

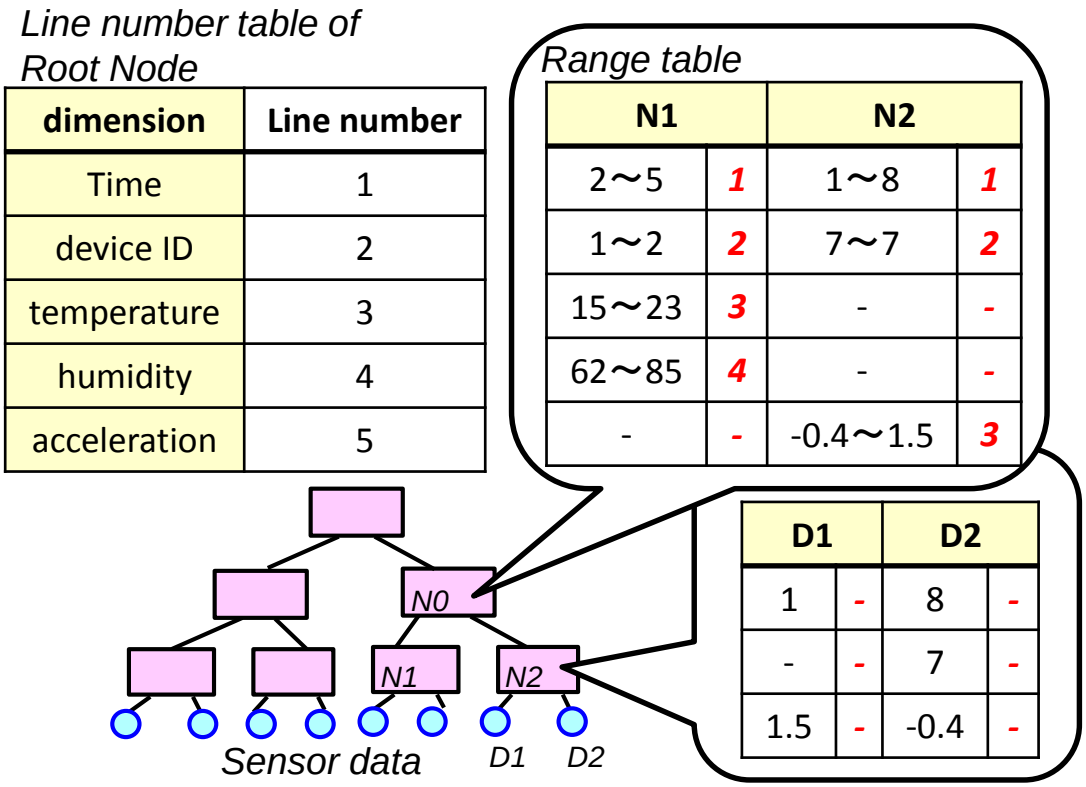


図 23　簡約化したデータ構造を用いた UBI-Tree

4.2.3 評価

提案手法 UBI-Tree の有用性を確認するため，大きく 2 種の評価実験を行った．まず 1 つ目に，UBI-Tree の特性とそれに伴う検索効率向上効果について確認した．2 つ目に，枯れた実装と言える PostgreSQL を比較対象として，想定センサデータセットに対する挿入処理・検索処理について，実時間上での比較評価を行った．以下，それぞれについて詳細に述べる．

4.2.3.1 UBI-Tree の特性と検索効率向上効果の確認

4.2.3.1.1 実験方法

提案手法は，次元の呪いの発生を抑えるために各ノードに含まれる次元種類数を減少させる．また各ノードの次元種類数と検索時にアクセスされる確率を減少させることにより検索効率を向上させる．これらのことを確認するため，データ挿入後の木構造の各高さにおけるノード内平均次元種類数と，検索時のアクセスノード数を測定した．検索プロセスの処

理負荷は概ねアクセスノード数に比例するため，アクセスノード数から検索効率を確認することができる．比較のため，UBI-Tree と同様に R-Tree [43]でも測定を行った．

実験には 2 種類のデータセットを用いた．データセット 1 は，データセット全体の次元種類数に対して多様なスキーマ群をもつデータセットを模擬したものである．各次元は各データ内に等確率でランダムに出現する．データセット 2 は逆に，データセット全体の次元種類数に対して最も少ないスキーマ群をもつデータセットを模擬したものである．データセット 2 では，データ種別によりスキーマが全く異なり，異なるスキーマ間で共通して出現する次元はない．実際のセンサデータセットにおいては，センサデータ種別により異なる次元が存在する一方で，時間や位置など，複数のセンサデータ種別に共通して出現する次元が存在し，どちらがどれほど多く存在するかは場合によって異なると考えられる．データセット 1 とデータセット 2 はその一方を 0%または 100%にしたケースであり，実際のセンサデータセットはそれらの中間的な性質をもつと考えられる．

データセット 1，2 はそれぞれ，全体で 700 次元を含み，10000 個のデータを含む．各データは 7 次元を含む．R-Tree は同種同数次元のデータセットを扱うため，R-Tree でインデキシングする場合には，各データが 700 次元全てをもつよう，存在しない次元を補完した．補完した次元の値は一律に 0 とした．R-Tree のノード分割アルゴリズムには Quadratic-Cost Algorithm を用いた．また UBI-Tree，R-Tree ともに，E_{max}，E_{min} の値はそれぞれ 10，5 とした．

データセット 1：

次元種類数は 700（a_i; i = 1, …, 700）とする．各次元はデータ内にランダムに出現するが，同じ次元が 1 つのデータに複数回出現することはない．各次元の値は 0～1000 からランダムに選ぶ．例を図 24 に示す．

[a_{698}=488, a_{552}=213, a_{417}=978, a_{179}=740, a_{80}=108, a_{110}=298, a_{671}=327]

[a_{575}=698, a_{525}=317, a_{104}=593, a_{191}=443, a_{302}=894, a_{446}=19, a_{637}=789]

[a_{56}=512, a_{646}=100, a_{617}=581, a_{37}=28, a_{152}=130, a_{641}=983, a_{461}=963]

図 24　データセット 1 の例

データセット 2：

次元種類数は 700，データ種別数は 100（a_i; i = 1, …, 100）とする．同じデータ種別の

データは同じ 7 次元をもつ（$a_{i_j}; j = 1, \cdots, 7$）．ある次元が複数のデータ種別のデータに出現することはない．各次元の値は 0～1000 からランダムに選ぶ．例を図 25 に示す．

[a_{37_0}=655, a_{37_1}=378, a_{37_2}=640, a_{37_3}=450, a_{37_4}=553, a_{37_5}=83, a_{37_6}=953]
[a_{21_0}=79, a_{21_1}=200, a_{21_2}=353, a_{21_3}=217, a_{21_4}=62, a_{21_5}=372, a_{21_6}=346]
[a_{37_0}=451, a_{37_1}=12, a_{37_2}=796, a_{37_3}=744, a_{37_4}=201, a_{37_5}=864, a_{37_6}=790]

図 25　データセット 2 の例

各データセットに合わせて 3 種類の検索条件セットを用いた．検索条件セット 1 はデータセット 1 用，検索条件セット 2-1 と 2-2 はデータセット 2 用である．各検索条件セットは 1000 個の検索条件を含み，各検索条件は 3 次元の範囲検索条件を含む（3 項を含む AND 条件）．

検索条件セット 1：

各検索条件について，データセット 1 に含まれる 700 次元から 3 つをランダムに選択し，検索条件に用いる．0～1000 からランダムに 2 つの値を選択し，小さい方を範囲検索条件の始点，大きい方を範囲検索条件の終点とする．例を図 26 に示す．

[a_{257}=587~768, a_{328}=373~661, a_{437}=271~990]
[a_{263}=16~154, a_{611}=419~746, a_{127}=238~795]
[a_{640}=358~757, a_{546}=122~190, a_{697}=429~570]

図 26　検索条件セット 1 の例

検索条件セット 2-1：

各検索条件について，データセット 2 の 100 種のデータ種別からランダムに 1 つのデータ種別を選択し，さらにそのデータ種別が含む 7 つの次元からランダムに 3 つの次元を選択し，検索条件に用いる．値範囲の決め方は検索条件セット 1 と同様である．例

を図 27 に示す.

[a_{21_4}=19~746, a_{21_1}=641~682, a_{21_5}=278~880]
[a_{76_1}=394~405, a_{76_3}=504~792, a_{76_5}=42~708]
[a_{61_5}=50~948, a_{61_3}=129~864, a_{61_6}=363~413]

図 27　検索条件セット 2-1 の例

検索条件セット 2-2：

各検索条件に用いる次元の選択方法は検索条件セット 2-1 と同様である．値範囲は 0~1000 に固定する．例を図 28 に示す.

[a_{21_4}=0~1000, a_{21_1}=0~1000, a_{21_5}=0~1000]
[a_{76_1}=0~1000, a_{76_3}=0~1000, a_{76_5}=0~1000]
[a_{61_5}=0~1000, a_{61_3}=0~1000, a_{61_6}=0~1000]

図 28　検索条件セット 2-2 の例

4.2.3.1.2 実験結果と考察

データセット 1，データセット 2 を挿入した後の，木の各高さ（lv）の平均次元種類数をそれぞれ図 29，図 30 に示す．これらの図において，lv0 はリーフノードの高さであり，lv4 はルートノードの高さである.

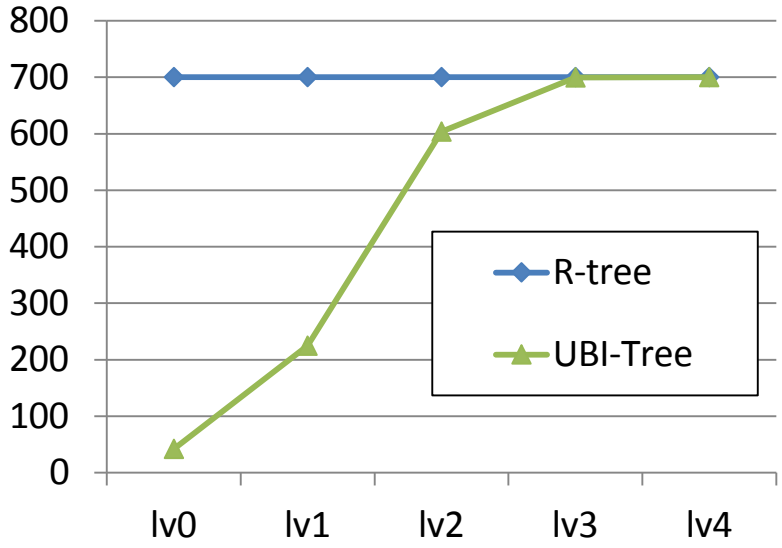


図 29　データセット 1 挿入後の次元種類数

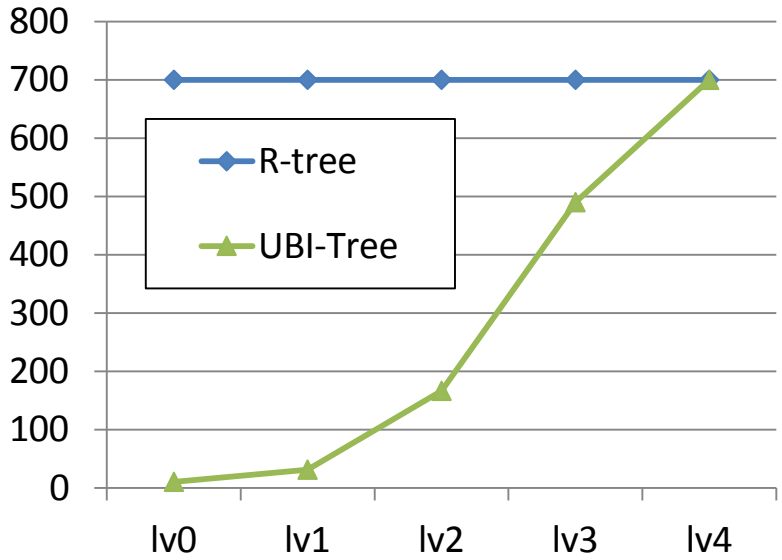


図 30　データセット 2 挿入後の次元種類数

図 29，図 30 から分かるように，R-Tree ではリーフノードでも次元種類数は大きい．これは，R-Tree における各データは存在しない次元も補完されているためである．一方，UBI-Tree の次元種類数は期待通りリーフノードに近づくにつれて減少することがわかる．図 29 と図 30 を比較すると，データセット 1 の次元種類数の減少度合いはデータセット 2 より

も小さいことが分かる．これは，データセット 1 のように各データに次元がランダムに出現する場合，次元種類数が減少するようにデータを分類することが本質的に困難なためと考えられる．

データセット 1 に対して検索条件セット 1 で検索した場合のアクセスノード数を図 31 に示す．データセット 2 に対して検索条件セット 2-1，2-2 で検索した場合のアクセスノード数を図 32 に示す．アクセスノード数は，検索条件セットに含まれる各検索条件におけるアクセスノード数の平均値である．

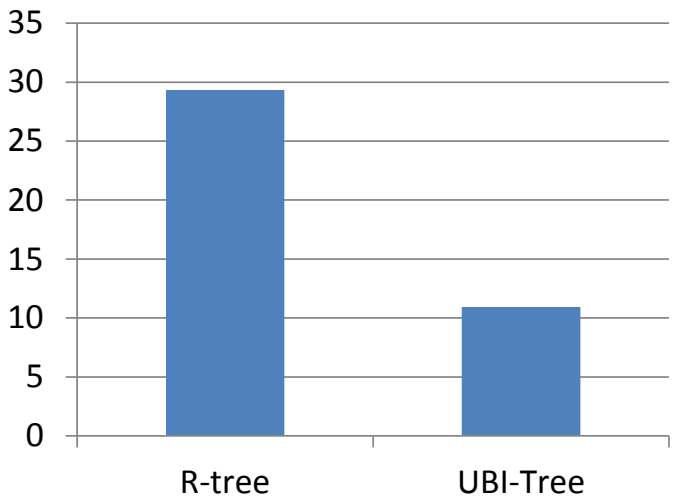


図 31　データセット 1 に対する検索条件セット 1 のアクセスノード数

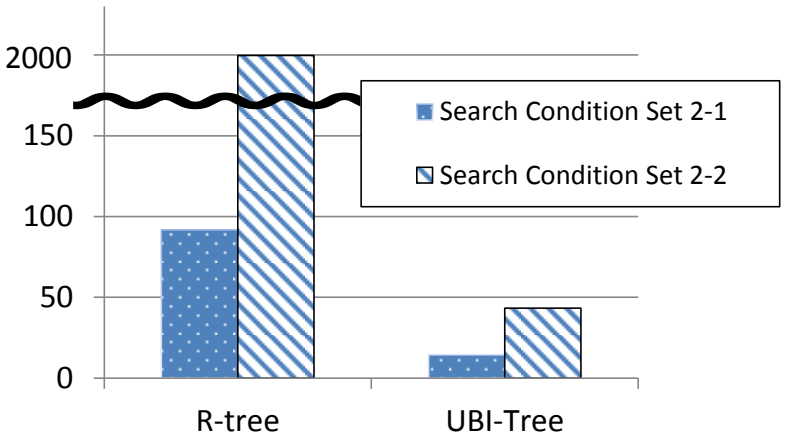


図 32　データセット 2 に対する検索条件セット 2-1，2-2 のアクセスノード数

図 31，図 32 から分かるように，UBI-Tree を用いた場合のアクセスノード数は R-Tree を用いた場合に比べてずっと小さい．UBI-Tree の検索効率は，R-Tree よりも良いと言え，これは UBI-Tree が各ノードの次元種類数の増加を防いでいるためと考えられる．これにより次元種類によるアクセスノードの絞込みが可能となり，また次元の呪いの発生が抑制されるため値範囲によるアクセスノードの絞込みも合わせて可能となっている．

図 32 から，R-Tree に対する UBI-Tree の改善度合いは検索条件セット 2-1 よりも検索条件セット 2-2 においてより大きいことが分かる．値範囲の広い検索条件で検索する場合，値範囲によるアクセスノードの絞込みができないため，検索負荷は大きくなる．しかしながら，UBI-Tree では次元種類によりアクセスノードを絞り込むことができるため，検索効率をある程度維持することができていると考えられる．R-Tree に対し UBI-Tree は，例えば「温度次元を含むデータ」のように広い値範囲の検索条件に対して特に改善効果が高いと言えよう（ある次元を含むという条件は，その次元の値範囲が－∞~∞であるという範囲条件と考えることができる）．

4.2.3.2 想定センサデータセットに対する挿入・検索処理時間の確認

4.2.3.2.1 実験概要

ここまでの評価により，従来技術に対して UBI-Tree は，多様なセンサデータセットに対して有効であることを確認した.すなわち，複数のセンサデータ種別に共通して出現する次元が多い場合にも少ない場合にも，程度の差はあるものの木構造の中でリーフノードに近づくにつれて次元種類数を減少させることができ，検索時のアクセスノード数を減少させることができる．

しかしながら，UBI-Tree のデータ構造は，B-Tree や R-Tree などと比べて扱う次元の種類や個数が動的に変化するため，挿入 / 検索時の処理内容もそれに応じて煩雑なものとなる．実用化に向けては，アクセスノード数に基づく理論的な検索効率確認だけではなく，高速な挿入 / 検索処理を行える実装が可能であることを確認することが重要である．そこで，枯れた従来技術の実装の 1 つとして PostgreSQL を比較対象とし，挿入 / 検索処理の実時間上での性能確認を行った．

まず，そのための準備として，UBI-Tree のパラメータ，すなわち E_{max}，E_{min} を変化させた場合の，挿入 / 検索性能とメモリ消費量を測定した．これによりパラメータに対する UBI-Tree の性能やメモリ消費量への影響を確認するとともに，最適なパラメータセッティングを明らかにした．また，4.2.2.4 で述べた簡約化したデータ構造を用いた場合，メモリ消費量は減少すると期待されるものの，ノードを辿る度に行番号を調べる他，ノード分割時の範囲表分割処理における行番号メンテナンスのオーバーヘッドが発生するため，特に挿入性能の劣化が心配される．そこでデータ構造を簡約化した場合，しない場合における，UBI-Tree のメモリ消費量と挿入性能をそれぞれ測定した．これにより，データ構造の簡約

化による性能への影響を確認するとともに，どちらが良いデータ構造であるかを明らかにした．その後，最適なパラメータ，最適なデータ構造を用いた UBI-Tree と，PostgreSQL それぞれにおける挿入 / 検索性能を測定した．

実験には，なるべく現実的な状況を模擬するため，これまで実証実験等で扱った種々のセンサデータに基づき設計した，表 3 に示すデータセット 3 を用いた．例を図 33 に示す．データセット 3 は，50 種のデータ種別が混在している．subject と type の 2 つの次元の値の組み合わせが 50 種類存在し，各データ種別に対応している．個々のデータは 10～13 個の次元をもち，そのうち 8 つの次元はどのデータ種別にも共通して出現する次元（共通次元），残りの次元はデータ種別固有の次元（非共通次元）である．データセット 3 は，全次元が全データに出現しうるデータセット 1 と，全て非共通次元であるデータセット 2 の中間的な性質をもつと考えられる．同じデータ種別でも非共通次元の数はランダムに 2～5 個とし，多様なスキーマが混在するようになっている．例えば同じ加速度センサでも，2 次元加速度センサと，3 次元加速度センサが存在し，それぞれ非共通次元の数が異なる様子を模擬している．全体では 258 次元を含むデータセットであり，データ 1 つの平均サイズは 293byte である．なお，UBI-Tree の実装は C++で行い，Boost ライブラリを用いた．

表 3　データセット 3

Dimension		Type	Value	Meaning
Common Dimension	_cdate	int	Increase by 1 from 1000000001	Inert time
	_userial	int	1	user ID
	address	String	Format is “0x000…12345 (48 figures.) Last 5 figures are randomly selected from 10001 to 20000.”	device ID
	date	int	Same as _cdate	Measurement time
	position	int	Normal distribution with mean of 5000 and standard deviation 1500	Measurement position (latitude)
	position	int	Normal distribution with mean of 5000 and standard deviation 1500	Measurement position (longitude)
	subject	String	One of ten types of strings*1	Sensor type
	type	String	One of five types of strings defined by subject value*1	Message type
Uncommon Dimension*2	b_(type)_(subject)	int	Normal distribution with mean ranging from 5000 to 29500 and standard deviation 1500	Dimensions specific to data type
	h_(type)_(subject)			
	n_(type)_(subject)			
	u_(type)_(subject)			
	z_(type)_(subject)			

*1 The strings are randomly selected in advance from /usr/share/dict in Linux.
*2 (type) and (subject) are the mean values of type and subject. The number of uncommon dimensions are randomly selected from 2 to 5.

[_cdate=i:1000000001, _userial=i:1, address=s:0x00000000000000000000000000000000000000000000018518, b_decompressing_chariotway=i:663, date=i:1000000001, positionx=i:9840, positiony=i:8045, subject=s:chariotway, type=s:decompressing, u_decompressing_chariotway=i:8239, z_decompressing_chariotway=i:8061]

[_cdate=i:1000000002, _userial=i:1, address=s:0x00000000000000000000000000000000000000000000013315, b_saccharometabolism_glove=i:5081, date=i:1000000002, h_saccharometabolism_glove=i:20, positionx=i:6029, positiony=i:1660, subject=s:glove, type=s:saccharometabolism, u_saccharometabolism_glove=i:7719]

[_cdate=i:1000000003, _userial=i:1, address=s:0x00000000000000000000000000000000000000000000015511, date=i:1000000003, positionx=i:486, positiony=i:1023, subject=s:nonhomaloidal, type=s:reverentially, u_reverentially_nonhomaloidal=i:7656, z_reverentially_nonhomaloidal=i:63]

図 33　データセット 3 の例（次元は辞書順に整列されている）

4.2.3.2.2 UBI-Tree パラメータ変更に関する実験

E_{max}，E_{min} を E_{max}=2*E_{min} という条件下で変化させた場合の，50 万データ挿入に要する時間と，50 万データ挿入後のメモリ消費量，50 万データ挿入後に行うヒット数 1 の検索 1000 回に要する時間を図 34 に示す．検索条件には，ヒット数が 1 となるよう値範囲を調整した positionx，positiony の 2 次元範囲検索を用いた．また UBI-Tree は Intel Core 2 Quad 3.00GHz，4GB Memory，CentOS 6.0 の計算機上で動作させた．図 34 から分かるように，挿入時間，検索時間ともに，[E_{max}, E_{min}]の組み合わせが[10, 5]，[15, 7]，[20, 10]の場合に最小，つまり最も良い性能が得られている．ノード内のエントリ数が少なすぎると，木が大きくなり枝を辿る手間が増えてしまうが，逆にノード内のエントリ数が多すぎるとノード内の処理負荷が高くなるものと思われる．メモリ消費量は[30, 15]，[40, 20]の場合に最小となったが，[10, 5]，[15, 7]，[20, 10]においても最小に近い値となっている．また，E_{max}=2*E_{min} という条件を外し，[20, 5]で計測したところ，挿入性能，検索性能，メモリ消費量ともに悪化することが確認された．総合して，[20, 10]が最も良いパラメータセッティングの 1 つであると考え，以降の実験はこのセッティングを用いて行った．

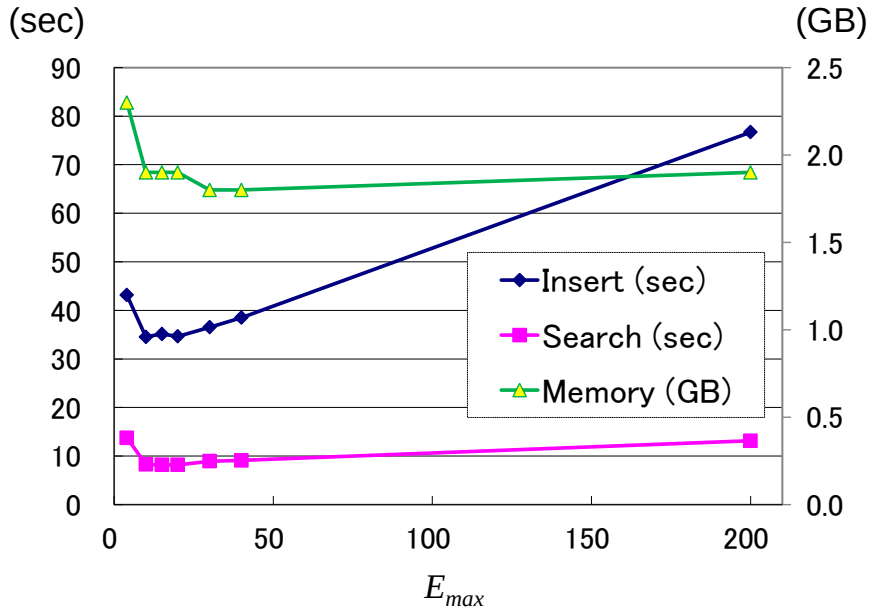


図 34　UBI-Tree パラメータの影響

4.2.3.2.3 データ構造簡約化に関する実験

簡約化したデータ構造を用いた場合，簡約化しないデータ構造を用いた場合における，UBI-Tree のメモリ消費量とデータ挿入に要する時間をそれぞれ図 35，図 36 に示す．このとき，UBI-Tree は Intel Core 2 Duo 3.16GHz，3.7GB Memory，CentOS 6.0 の計算機上で動作させた．図 35 から分かるように，等量のデータを挿入した場合，データ構造を簡約化しない場合に比べ，簡約化した場合には，期待通りメモリ消費量が圧倒的に少なく済むことが分かる．簡約化しない場合，データ数が 10 万件程度のあたりでメモリ消費量は頭打ちとなっているが，スワップが発生し始めているものと考えられる．また図 36 から分かるように，データ構造を簡約化した場合，予想に反して高速性が得られることが分かる．行番号メンテナンスのオーバーヘッドよりも，多くのメモリ確保 / メモリコピーなどに要する時間が省かれたことによる性能改善効果が大きいものと考えられる．総合して，データ構造を簡約化した場合が良いと考え，以降の実験は簡約化したデータ構造を用いて行った．

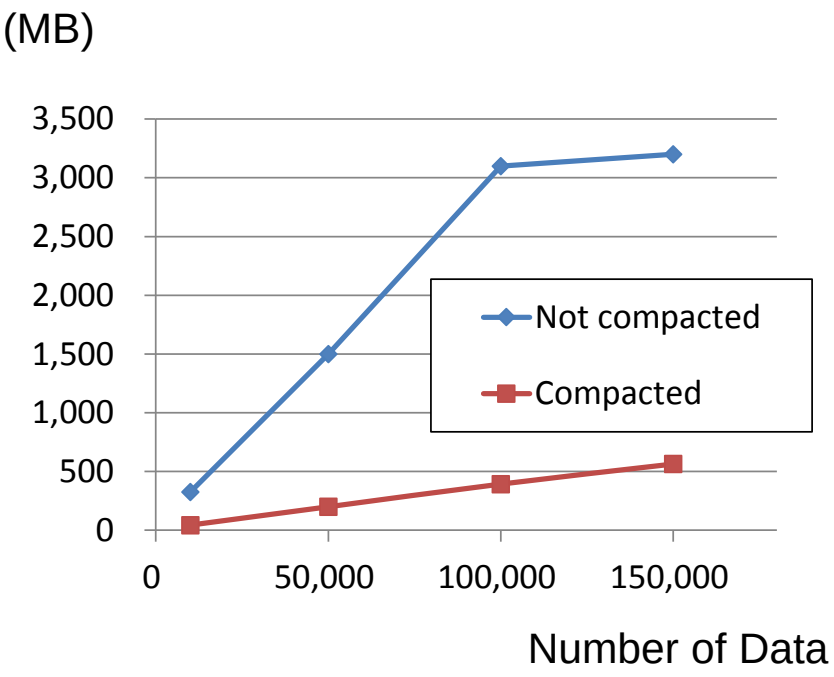


図 35　各データ構造を用いた場合のメモリ消費量

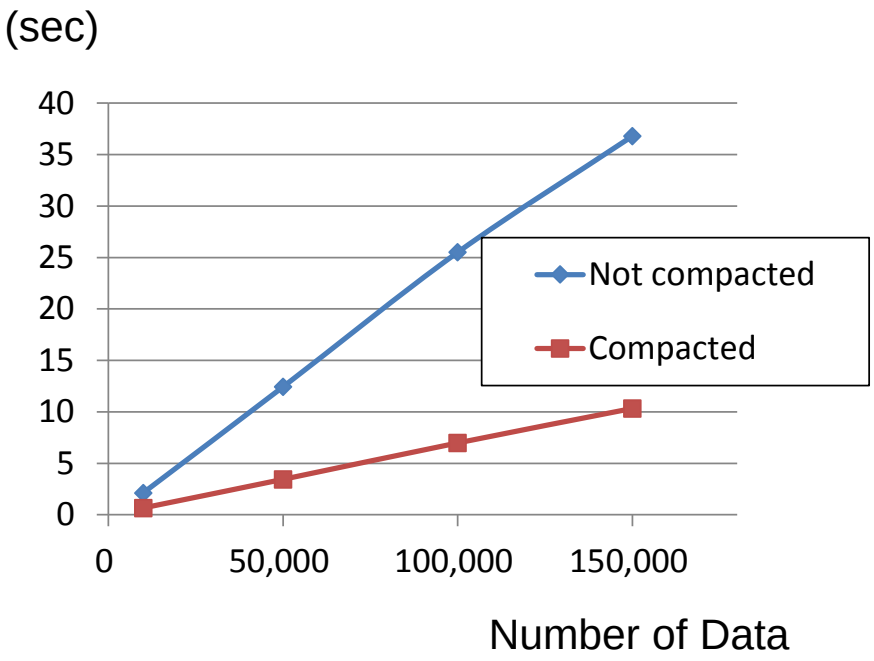


図 36　各データ構造を用いた場合の挿入性能

4.2.3.2.4 PostgreSQL との挿入/検索性能比較実験

これまでの評価結果から，最適なパラメータとして[E_{max}, E_{min}] = [20, 10]，また最適なデータ構造として簡易化したデータ構造を用いた UBI-Tree と，PostgreSQL それぞれの挿入 / 検索性能を測定した結果を図 37～図 43 に示す．このとき UBI-Tree や PostgreSQL は Intel Core 2 Duo 3.16GHz，3.7GB Memory，CentOS 6.0 の計算機上で動作させた．PostgreSQL のバージョンは 8.4.9 である．PostgreSQL を用いる場合，50 種のデータ種別のデータは通常それぞれ別のテーブルに挿入するものと思われる．ただし，共通次元で検索する場合，複数テーブルから検索しなければならないため，性能的に不利になる可能性がある．例えば x 個のデータを y 分割し，y 個の B-Tree を用いて検索する場合には計算量オーダは $O(ylogx)$となるが，分割せずに 1 つの B-Tree を用いて検索する場合には $O(logxy)$となる．そこで基本的に PostgreSQL を用いる場合には，50 種のデータ種別ごとにテーブルを分けた場合と，全データ種別を 1 つのテーブルに入れる場合の 2 つのパターンで測定を行った．また PostgreSQL で使用するインデックスパターンとして，インデックスを全く付与しないパターン（インデックスなし），全カラムにデフォルトのインデックスを付与するパターン（B-Tree パターン），date，positionx，positiony の 3 次元には汎用検索木 GiST インデックス [55]の cube モジュールを用い，残りの全カラムにはデフォルトのインデックスを付与するパターン（GiST パターン）の 3 つを用いた．なお，UBI-Tree は，挿入時にメモリ上でインデキシングを行うとともに，ジャーナルファイルへの書き込みを行っているが，性能を重視するために同期書き込みは行わず，1 秒に 1 回 fsync を実行する．そこで PostgreSQL のパラメータ設定においても，fsync=off とした．他はデフォルトのままである．このとき，PostgreSQL のブロックサイズは 8192byte であった．

図 37，図 38 に UBI-Tree と PostgreSQL における，データ挿入に要する時間を示す．図 37 は PostgreSQL において 50 テーブルを用いた場合，図 38 は 1 テーブルを用いた場合のデータ挿入時間である．比較しやすいように，UBI-Tree のデータ挿入時間は両方の図に記載してある．これらの図から，PostgreSQL の挿入性能は，50 テーブルを用いる場合に比べ，1 テーブルを用いた場合に大きく劣化することが分かる．またインデックスなしの場合に比べ，GiST パターン，B-Tree パターンの順で性能劣化することが分かる．インデックスの本数が多いほど，挿入性能が劣化していると思われる．またインデックスを付与しない場合，所要時間は挿入データ数に比例するが，GiST パターンや B-Tree パターンでは挿入データ数に対して線形以上の速度で所要時間が増加してしまうことが分かる．データ数が増加すると木構造も大きくなり，挿入処理のコストが高くなるためと考えられる．これに対し UBI-Tree の挿入性能は，PostgreSQL の最も性能が出る場合，すなわち 50 テーブル，インデックスなしの場合の挿入性能より優れることが分かる．PostgreSQL においては SQL 文の複雑な構文解析を含むなど，挿入処理内容自体がやや異なるために単純に比較することはできないが，UBI-Tree1 つのインデックスを付与するオーバーヘッドは十分に小さいと言えよう．

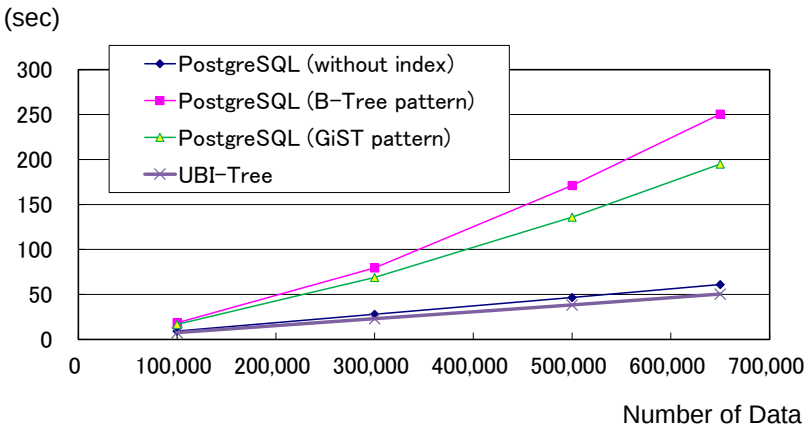


図 37　挿入性能（PostgreSQL は 50 テーブル使用）

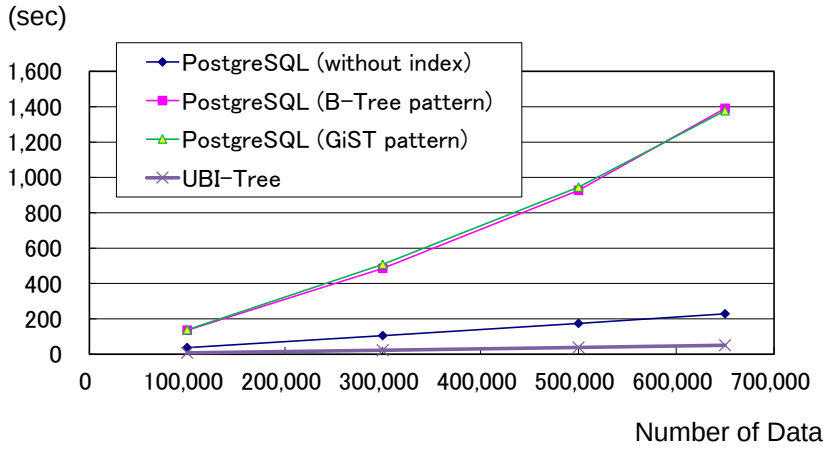


図 38　挿入性能（PostgreSQL は 1 テーブル使用）

図 39～図 43 は，UBI-Tree と PostgreSQL の検索性能を示す．図 39～図 42 は，検索条件に含まれる次元数を 1 に固定し，検索にヒットするデータの割合（ヒット率）を変化させた場合の検索時間である．検索時の蓄積データ数は 65 万件とした．また PostgreSQL は 50 テーブルを用いた場合の検索性能である．1 テーブルを用いた場合，共通次元，0.1%ヒット率の検索において 50 テーブルの場合よりも若干良い検索性能を示したが，他のケースでは 50 テーブルの場合の方が良い検索性能を示したため，50 テーブルの場合のみを掲載した．図 39，図 40 は共通次元（positionx）1 次元で検索した場合の検索時間，図 41，図 42 は非共通次元 1 次元で検索した場合の検索時間である．表 3 に記載したように，positionx，非共通次元ともに値は平均 5000，標準偏差 1500 の正規分布に従う．図 39，図 41 は検索条件として指定する値範囲を正規分布の中央付近（平均値を中央とする値範囲）とした場合，

図 40，図 42 は正規分布の端の方（それより外側に存在する値の数が全体の 5%となる値を中央とする値範囲）とした場合の結果である．各測定においては，目的のヒット率となるよう予め値範囲の調整を行った．

図 39〜図 42 より，1 次元の検索において，概ねどの場合にも PostgreSQL に対し UBI-Tree は優れた性能を示すことが分かる．1 次元の検索では，本来 B-Tree は UBI-Tree より効率的な検索ができるはずであるが，前述したように PostgreSQL においては SQL 文の構文解析などデータベース管理システムとしての機能を実現する各処理を含むため，このような結果となったと考えられる．また本結果より，本オーバーヘッドはヒット率が大きい場合に大きくなり，このとき UBI-Tree は本オーバーヘッドに対し十分に高速であると言える．逆にヒット率が十分に小さい場合には，B-Tree を用いた PostgreSQL の方が UBI-Tree よりも良い性能となることが分かる．

図 43 は，ヒット率を固定し，検索条件に含まれる次元数を変化させた場合の検索性能である．このとき，なるべく PostgreSQL に有利な条件となるよう，ヒット率を 0.01%と小さい値に固定した．またデータ種別は 1 種のみを用い，PostgreSQL は 1 テーブルに全データを格納することとした．これにより 50 テーブルから結果を集約する処理を不要とし，またカラム数の多いスパースな 1 テーブルから検索することも不要とした．UBI-Tree が得意とするスパース性・多様性を含まないデータセットとも言えるが，多次元をひとまとめに検索する UBI-Tree の多次元インデックスとしての効果を確認することができる．蓄積データ数は 10 万件とした．検索条件には次元を poxitionx, positiony, date の順で含めていくことで次元数を増やしていき，その後は非共通キーを 1 つずつ含めていくこととした．また多次元範囲検索において，各次元の値範囲が検索結果を絞り込む割合は等しくなるようにした．例えば 2 次元範囲検索の場合，positionx，positiony それぞれによるヒット率は 0.1%ずつとし，AND をとることで最終的なヒット率が 0.01%となるようにした．また PostgreSQL においては，B-Tree パターンを用いた．

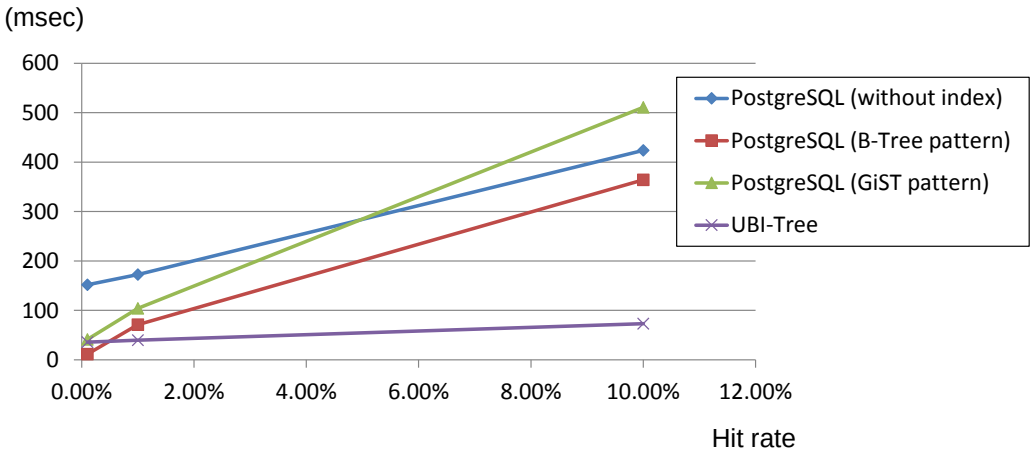


図 39　検索性能（共通次元，中央）

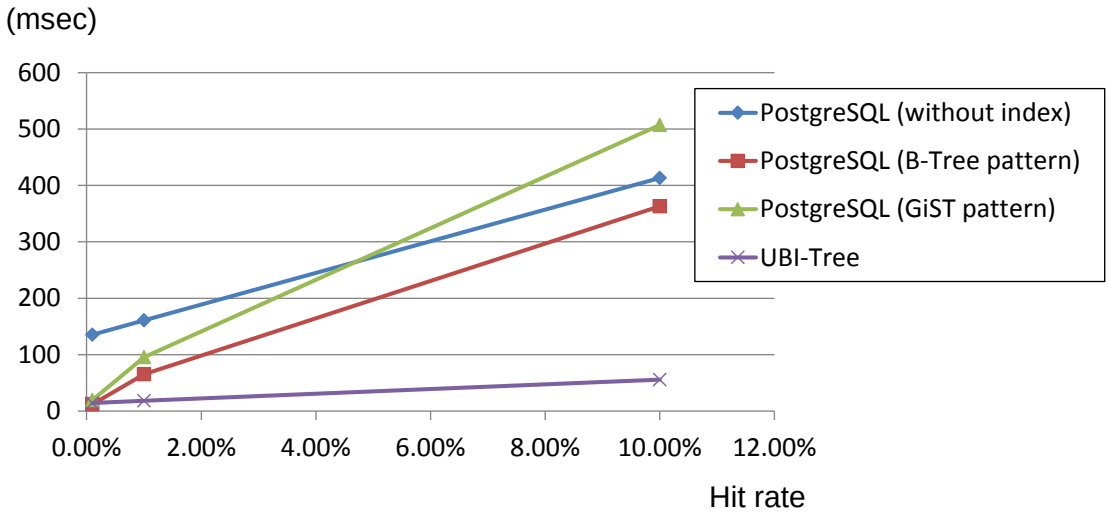


図 40　検索性能（共通次元，端）

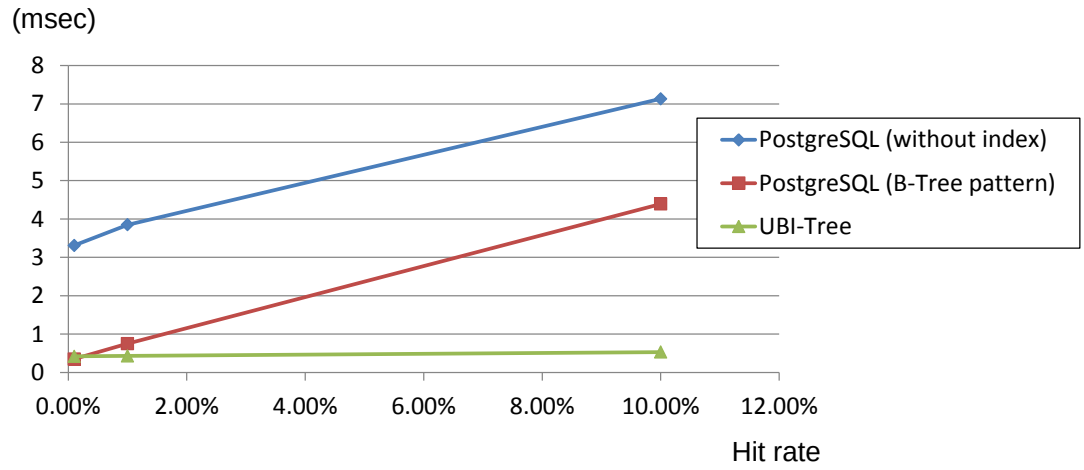


図 41　検索性能（非共通次元，中央）

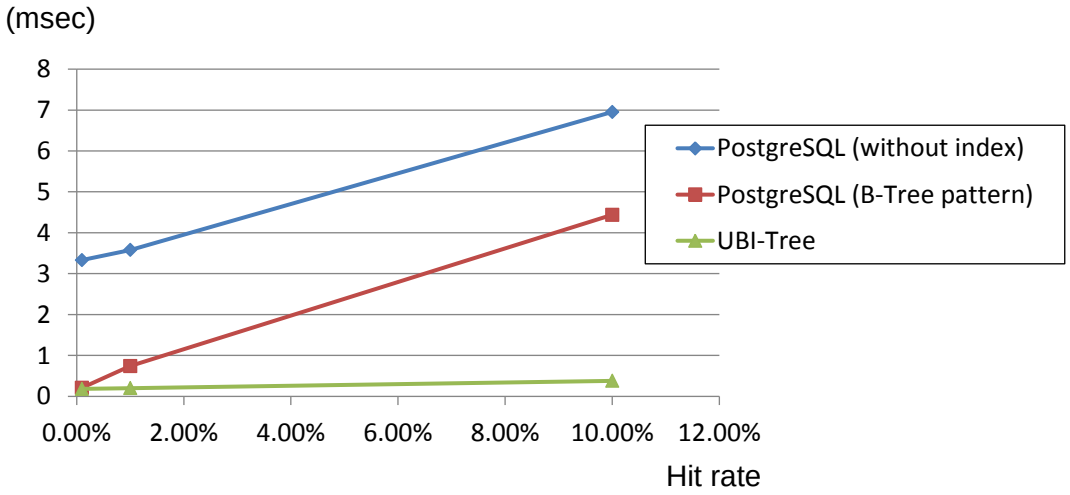


図 42　検索性能（非共通次元，端）

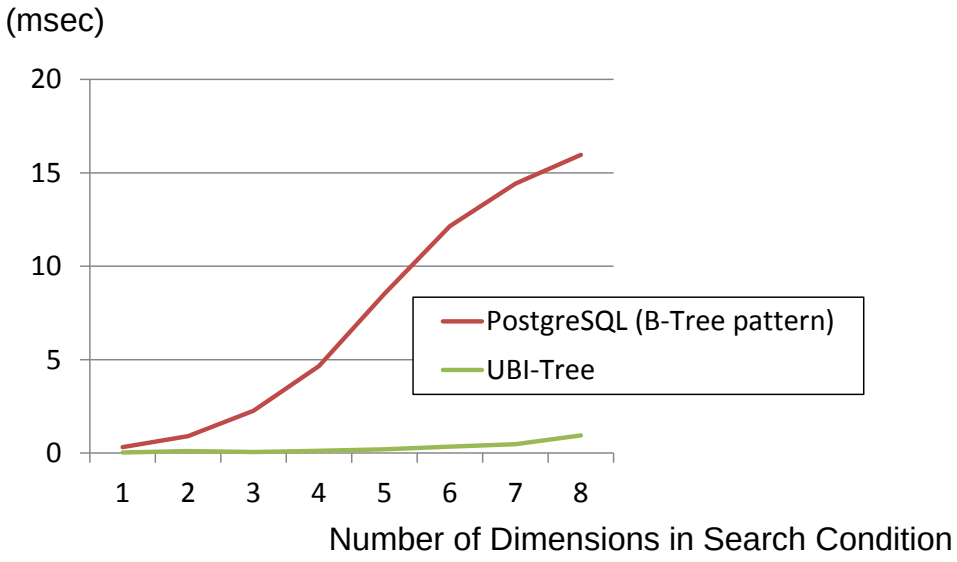


図 43　検索性能（多次元）

図 43 より，PostgreSQL は検索条件に含まれる次元数が増えるに従い，検索時間が増加するが，UBI-Tree は多次元検索においても，検索性能を維持できることが分かる．1 次元の検索では UBI-Tree, PostgreSQL ともに同程度の性能であるから，前述した PostgreSQL のオーバーヘッドは本実験では十分に小さいと考えられ，多次元検索における性能の差異はアルゴリズムの差異によるものと考えられる．PostgreSQL の EXPLAIN コマンドにより実行プランを調べたところ，2 次元検索の場合には 2 つのインデックスを用いて結果をマージし，3 次元以上の検索の場合には 1 つのインデックスを用いて中間解を取得し，その後フィルタリングを行うことが分かった．PostgreSQL では，多次元範囲検索において 1 次元だけで検索結果を十分に絞り込めない場合，マージやフィルタリングによる処理コストが大きくなることが分かる．一方，UBI-Tree では，検索条件に含まれる全次元を考慮して絞込みを行っていくことにより，効率的に検索することができていると考えられる．データが多様になり，PostgreSQL において複数テーブルを用いる場合には，この性能差はさらに広がると考えられる．

以上より，UBI-Tree は PostgreSQL に比べ，多様なデータに対して 1 つのインデックスだけで多様な検索を効率化することができるために挿入性能の面でメリットがあり，さらに多次元範囲検索において検索条件に含まれる次元数が増えた場合にも，期待通り高い検索性能を維持できることが確認できたと言える．

4.2.4 UBI-Tree インデキシング技術まとめ

4.2 ではまず，スキーマレス検索を効率化する必要性について述べた．次に，スキーマレス検索のためのインデキシング技術 UBI-Tree について提案した．さらに，実験により，従来技術に比べ，UBI-Tree はよりスキーマレス検索を効率化できることを示した．

4.3 UBI-Tree インデキシング技術の動的負荷適応手法の検討

4.2 では，スキーマレス検索を効率化するインデキシング技術 UBI-Tree を提案した．R-Tree [43]など多次元範囲検索のための従来のインデキシング技術は，同種同数次元を含む均一なデータを前提とする．このため異種異数次元を含む多様なデータにおいてはデータ間の距離を決定できず，データを分類することができない．これに対し UBI-Tree は，各データが含む値だけでなく次元の種類そのものの違いを考慮することで，多様なデータ集合を 1 つの木構造に分類格納することができる．またこのとき，蓄積されるデータの特性からクエリを予測し，頻繁に用いられると予測されるクエリほど効率的に処理できるようデータを分類することにより，平均レスポンスやスループットを向上させる．例えば，温度や湿度の次元に比べて降雨量，計測時間，計測位置（緯度 / 経度）の 4 次元の方が蓄積データ内に多く含まれるのであれば，当該 4 次元を用いたクエリが多用されると予測し，温度や湿度に比べて当該 4 次元を重視したデータ分類を行う．

しかしながら，予測に基づいてインデキシングを行うため，予測したクエリと真のクエリが異なる場合には性能が劣化するという問題がある．例えばもし降雨量分布を表示するアプリケーションだけでなく，豪雨監視アプリケーションが新たに出現し，当該アプリケーションがより高い頻度で降雨量, 計測時間の 2 次元クエリによる検索を行い始めた場合には，降雨量，計測時間，計測位置（緯度 / 経度）の 4 次元のうち，当該 2 次元を特に重視した方が平均レスポンスやスループットの向上に繋がると考えられる．

そこで本節では，実行された各種クエリの履歴の統計情報を動的に取得し，当該統計情報に基づき，実際により頻繁に用いられるクエリ（FQ; Frequently-used Query）の効率化を重視してインデキシングを行う改良版 UBI-Tree（Load-Adaptive UBI-Tree）を提案する．また提案手法により，頻度の低いクエリ（AQ; Ad-hoc Query）の性能は同程度か若干劣化するものの，FQ の性能が向上し，データストアの平均レスポンスやスループットが改善し得ることを示す．

以下，4.3 の構成は次の通りである．4.3.1 で従来の UBI-Tree や関連技術における問題点

を述べ，4.3.2 で負荷追従性を実現する UBI-Tree の改良手法を提案する．4.3.3 で評価実験内容について説明し，4.3.4 でまとめる．

4.3.1 従来技術とその問題

4.3.1.1 UBI-Tree インデキシング技術の問題

UBI-Tree は R-Tree をベースとした木構造インデキシング技術である．図 19 に示したように，R-Tree と異なり，UBI-Tree は多様なデータをインデキシングするため，各ノードが互いに異なる次元を保持し得る．UBI-Tree はデータ分類の悪さを示す 2 種類の指標を用い，これらの指標がなるべく増加しないように多様なデータを各ノードに分類することでスキーマレス検索を効率化する．

1 つ目の指標は各ノードの「次元種類数」である．次元種類数が増加しないようにデータを分類することで，同じ次元をもつデータが同じノードに分類される．これにより，リーフノードに近づくにつれ各ノードの次元種類数を急激に減少させることが可能となる．データが高次元になるとデータ間の距離の比が漸近的に 1 に近づき，データのクラスタリングが本質的に困難になる．このような現象は次元の呪いと呼ばれ，検索効率低下の大きな原因となる．次元種類数を減らすことで次元の呪いを抑え，検索効率を向上させることができる．

2 つ目の指標は「各ノードが検索時にアクセスされる確率」である．これが増加しないようにデータを分類することで，検索時のアクセスノード数が減少し，効率的な検索が可能となる．ノード N が検索時にアクセスされる確率 P は次式で求める（再掲）．

$$P = \sum_{S \subseteq U} P_c(S) \prod_{D \in S} P_N(D) \tag{6}$$

ただし，U は蓄積データ内に存在する次元の全組合せを表す．$P_c(S)$は検索条件に次元集合 S が用いられる確率であり，$P_N(D)$は検索条件に次元 D が用いられた場合に，ノード N がその検索によりアクセスされる確率を示す．式 (6) は，何らかの検索が行われた場合に，ノード N がアクセスされる確率の期待値を求める式となっている．$P_c(S)$は次式で求める（再掲）．

$$P_c(S) = \frac{1}{M} \sum_{t_s \in T_s} \frac{1}{2^{V_{t_S}} - 1} \tag{2}$$

ただし，T_s は次元集合 S を含む蓄積データの部分集合を表し，t_s は T_s の要素である．またV_{t_s}

は t_s に含まれる次元種類数であり，M は蓄積データ数を表わす．式 (2) は，蓄積データに多く出現する次元ほど検索条件として多く用いられやすいという仮定に基づき，検索条件に次元集合 S が用いられる確率 $P_c(S)$を蓄積データの統計情報から推定する式である．

これら 2 つの指標は，次元の数や種類が異なるデータが混在した場合にも定義可能であり，UBI-Tree は多様なデータを 1 つの木構造にインデキシングすることができる．UBI-Tree はまず「次元種類数」により各データ種別のデータをほぼ別々の部分木に分類し，さらに各部分木内において「各ノードが検索時にアクセスされる確率」により同じデータ種別内でも特に値が近いデータ同士を同じノードに分類する．木構造インデックスにおいて検索効率を向上させるためには，検索時にうまく枝刈りすること，つまりアクセスするノードをいかに絞り込めるかが重要である．UBI-Tree は，クエリに合致するデータを含むノードを次元の種類と値により効率的に絞り込むことで，多様なデータに対するスキーマレス検索を効率化する．

しかしながら，以上述べた UBI-Tree の基本的アルゴリズムだけでは，2 つの問題により検索効率の低下を招く恐れがある．

問題 1:「検索条件に次元集合 S が用いられる確率 $P_c(S)$」を蓄積データの統計情報から精度良く予測することは難しい

蓄積データの統計情報から，当該確率をある程度予測することはできるが，その精度には限界がある．蓄積データが同じでも，アプリケーションが変われば検索条件は変化し得る．蓄積データから $P_c(S)$を予測するのは，蓄積データからアプリケーションを予想するのと同様に困難である．このため予測と実際との乖離が発生し，検索効率が低下してしまう場合がある．

問題 2:各ノードの次元種類数を減らすようデータを分類する方法が常に検索効率の向上に繋がるとは限らない

多様なデータを次元種類により分類すると，検索条件に含まれる次元の種類によりアクセスすべきノードは絞り込まれ，検索効率は向上する．しかしながら，蓄積データのほとんどに共通的に含まれる次元のみで検索する場合には，次元種類によりアクセスすべきノードを絞り込むことがほとんどできない．このようなクエリが多い場合には，同じ種類の次元をもつかどうかよりも，当該クエリに含まれる次元の値が近いかどうかを重要視してデータ分類した方が検索効率の向上に繋がる．

どちらの問題も，検索効率向上のためには負荷追従性，すなわち，実際の負荷に応じて適応的にデータ分類方法を変更する必要があることを示唆していると考えられる．

4.3.1.2 その他の従来技術とその問題

現在最も利用されているデータベースはリレーショナルデータベース（RDB）であろう．RDB においては，性能向上のために負荷に応じたチューニングが行われる．多様な多次元範囲検索に対するチューニングにおいて，インデックス設計としては例えば以下の 2 案が考えられるが，それぞれ問題がある．

インデックス設計 1：使用頻度の高いクエリの検索条件に含まれる次元にだけインデックスを付与

最も一般的なチューニングであると思われる．使用頻度の高いクエリ（FQ）は高速に実行できるものの，頻度の低いクエリ（AQ）は基本的に高速化されないという問題がある．

インデックス設計 2：全次元にインデックスを付与

FQ, AQ ともに高速に実行可能となるものの，インデックス構築の負荷が高く，データ挿入が遅くなってしまうという問題がある．

また UBI-Tree がベースとしている多次元インデキシング技術 R-Tree において，クエリに応じてデータ分類方法を変更する手法も提案されている [57]．しかしながら，R-Tree や k-d Tree [46]，あるいはその派生技術 [47] [48] [49]など従来の多次元インデキシング技術は，前述の通り，そもそも異種異数次元を含むデータをうまく扱うことができない．

4.3.2 提案手法「Load-Adaptive UBI-Tree」

4.3.2.1 基本的アイデア

以上述べた従来の問題を鑑み，実際の負荷に応じて適応的にデータ分類方法を変更する新しい UBI-Tree（Load-Adaptive UBI-Tree）を提案する．

新しい UBI-Tree は，蓄積データの統計情報だけでなく，クエリ履歴の統計情報を用いて「検索条件に次元集合 S が用いられる確率 $P_c'(S)$」を予測する．実際の負荷を考慮することで，基本的アルゴリズムのみの UBI-Tree（従来 UBI-Tree）に比べ，より高い精度の予測が可能となると考えられる．この予測値に基づいて「各ノードが検索時にアクセスされる確率 P'」を算出し，データ分類に用いることにより，検索効率を向上させる．またさらに，当該予測により，検索条件に用いられる次元の多くが蓄積データのほとんどに含まれる次元（共通次元）と判断される場合に，「各ノードが検索時にアクセスされる確率」のみをデータ分

類の指標として用いる．本論文ではこれを「次元種類数を無効化する」と表現する．これにより，更なる検索効率の向上を狙う．

以下，新しい UBI-Tree における挿入プロセスと検索プロセスについて，それぞれ詳細を述べる．

4.3.2.2 挿入プロセス

一般に，木構造をもつインデックスにおける挿入プロセス（Insert）は，ルートノードから開始し，挿入先子ノード選択アルゴリズム（ChooseSubtree）を用いて選択した子ノードを再帰的に辿り，最終的に辿り付いたリーフノードへデータを挿入する [55]．各ノードが含む子ノードまたはデータの数には制約が設けられており，最小値は E_{min}，最大値は E_{max} である．E_{max} を越えた場合には，そのノードを 2 つのノードに分割し，子ノードまたはデータは各ノードに分配する．ノードの分割はノード分割アルゴリズム（Split）に従って実行される．

UBI-Tree の挿入プロセスは，挿入先子ノード選択アルゴリズムとノード分割アルゴリズムにおいて，「次元種類数」と「各ノードが検索時にアクセスされる確率」をデータ分類の悪さを示す指標として用いる点に特徴がある．また従来の UBI-Tree に対し，4.3 で提案する新しい UBI-Tree は，「各ノードが検索時にアクセスされる確率」の算出にクエリ履歴を用い，また定期的に無効化判定アルゴリズム（InvalidateDecision）を実行して「次元種類数」を無効化するか否かを判定する点に特徴がある．

以下，各アルゴリズムについて述べる．

挿入先子ノード選択アルゴリズム（ChooseSubtree）：

[CS1] 次元種類数を無効化するか否かを示す「無効化フラグ」が OFF の場合，CS2 へ．ON の場合，CS4 へ

[CS2] 「次元種類数」の増加が最小となる子ノードを選択する

[CS3] CS2 で該当する子ノードが複数存在する場合には，さらにその中から P'の増加が最小となる子ノードを選択し，処理を戻す

[CS4] P'の増加が最小となる子ノードを選択し，処理を戻す

ノード分割アルゴリズム（Split）：

[SP1] 「無効化フラグ」が OFF の場合，SP2 へ．ON の場合，SP4 へ

[SP2] 分割後の 2 つのノードの「次元種類数」の和がなるべく小さくなるように分割する

[SP3]　SP2 で該当する分割方法が複数存在する場合には，さらにその中から分割後の 2 つのノードの P'の和がなるべく小さくなる分割方法を選び，分割を実行し，処理を戻す

[SP4]　分割後の 2 つのノードの P'の和がなるべく小さくなる分割方法を選び，分割を実行し，処理を戻す

提案手法において，検索時にアクセスされる確率 P'は次式により算出する．

$$P' = \sum_{S \subseteq U} P_c'(S) \prod_{D \in S} P_N(D) \tag{8}$$

$$P_c'(S) = \frac{P_c(S)(100 - A) + P_a(S)A}{100} \tag{9}$$

ただし，$P_a(S)$はクエリ履歴の統計情報に基づく予測値である．A は新たに導入したパラメータ「クエリ履歴有効度」であり，UBI-Tree を用いる環境に応じて 0～100 の値を設定する．検索負荷の変化が小さく，クエリ履歴内のクエリと同様のクエリが今後も実行されると期待できる場合には A の値を大きく設定することでクエリ履歴に基づく予測を重視する．逆に検索負荷の変化が大きく，クエリ履歴に基づく予測精度が期待できない場合には A の値を小さく設定する．$P_a(S)$は次式で求める．

$$P_a(S) = \frac{1}{Q} \sum_{q_s \in Q_s} \frac{1}{2^{V_{q_s}} - 1} \tag{10}$$

ただし，Q_s は次元集合 S を含む過去クエリの部分集合を表し，q_s は Q_s の要素である．また V_{q_s} は q_s に含まれる次元種類数であり，Q は蓄積データ数を表わす．式 (10) は，過去に用いられたクエリに多く含まれる次元ほど今後も検索条件として多く用いられやすいという仮定に基づき，検索条件に次元集合 S が用いられる確率を推定する式である．

また定期的に（例えば 100 万回挿入プロセスを実行する度に 1 回），以下の無効化判定アルゴリズムを実行する．

無効化判定アルゴリズム（InvalidateDecision）:

[ID1] $P_c'(S)$を重みとして，各次元集合 S が蓄積データに含まれる割合(%)の重み付き平均を算出する.

[ID2] ID1 で算出した重み付き平均が C を超える場合，共通次元で検索する確率が十分に高いと判定して無効化フラグを ON に設定し，そうでない場合には無効化フラグを OFF に設定する

C は新たに導入したパラメータ「共通次元判定割合」であり，0～100 の値を設定する．C を小さくとればとるほど共通次元での検索が十分になされていると判定されやすく，次元種類数が無効化されやすい.

4.3.2.3 検索プロセス

検索プロセス（Search）は一般の木構造インデックスと同じである．検索条件に含まれる全ての条件を満たす子ノードを再帰的に辿り，目的のデータを見つけ出す．ただし，新しい UBI-Tree においては，従来の UBI-Tree と異なりクエリ履歴の保存が必要となる.

4.3.3 評価

提案手法により，従来に比べ AQ の性能は同程度か若干劣化するものの，FQ の性能が向上するため，データストアの平均レスポンスやスループットが改善すると期待できる．提案手法の有効性を確認するため，2 種類の評価実験を行った．多様なセンサデータを含む想定センサデータセットの挿入性能測定と，使用頻度に偏りのある数種の想定クエリに対する検索性能計測である．このとき，提案手法においては新たに導入した 2 つのパラメータを何種類か変更して計測するとともに，比較対象として最も利用されている従来技術の 1 つである PostgreSQL でも同様に計測を行った.

4.3.3.1 実験条件

実験には，想定センサデータセットとして表 3 に示したデータセット 3 を用いた．挿入性能としては，50 万件のセンサデータを含むデータセット 3 を挿入するのに要する時間を計測した．また検索性能としては，頻度の高いクエリ（FQ）と頻度の低いアドホッククエリ（AQ）の実行に要する時間をそれぞれ計測した．提案手法の効果を確認しやすくするため，クエリ履歴には FQ のみが記録されている状態でデータを 50 万件挿入し，その後各種クエリに対する実行時間を計測した．提案手法は FQ を学習し，FQ に適したデータ分類を

行うこととなる．これにより，提案手法では従来に比べて FQ の実行が高速化され，逆にそうでない AQ に対しては性能が劣化すると予測される．FQ は＜address, date＞の 2 次元範囲条件をもつクエリ 1 種とし, AQ は＜date, positionx, positiony＞, ＜date, non-common dimension＞，＜positionx, positiony, uncommon dimension＞の 3 種の 2 ないし 3 次元範囲条件をもつクエリとした（＜X, Y＞は，X と Y の 2 次元範囲条件を表す）．FQ とした＜address, date＞は 2.2 で述べた検索パターン 2 に相当し，センサデータに対する最も基本的な検索条件であると考えられる．従来の水平統合型プラットフォーム [12] [58]においても＜address, date＞に相当する引数をとる API がデータ取得のための基本的な検索 API として提供されている．2.1 で述べた蓄電池管理システムにおいても，＜address, date＞は個々の蓄電池の SOC(State Of Charge)推定のために最も頻繁に用いられる検索条件である．一方，安全点検のために異常な温度となった蓄電池を検索する検索条件＜date, positionx, positiony, temperature＞等は，これに比べて低い頻度で用いられる．

計測パターンは表 4 に示す 8 パターンとした. 計測パターン 1～4 は UBI-Tree を用いた計測である．計測パターン 1（P1）は，A=0, C=100 であり，クエリ履歴を考慮せずまた次元種類数が必ず有効になるパラメータセッティングであるため, 従来の UBI-Tree と等価である．計測パターン 2（P2），計測パターン 3（P3）は，A の値がそれぞれ 50，100 であり，クエリ履歴を重視する度合いが順に強くなる．特に計測パターン 3 は蓄積データの統計情報を無視し，クエリ履歴のみを考慮したデータ分類を行う．また計測パターン 4（P4）は，A の値は計測パターン 2 と同じであるが，C=0 であり，次元種類数が必ず無効となるセッティングである．計測パターン 5～8（P5～P8）は PostgreSQL を用いた計測である．PostgreSQL を用いる場合，50 種のデータ種別のデータは通常それぞれ別のテーブルに挿入するものと思われる．ただし，4.2.3.2.4 で議論したように，共通次元で検索する場合，複数テーブルから検索しなければならないため，性能的に不利になる可能性がある．そこで PostgreSQL を用いる場合には，50 種のデータ種別ごとにテーブルを分けた場合（計測パターン 7,8）と，全データ種別を 1 つのテーブルに入れる場合（計測パターン 5,6）でそれぞれ測定を行った．また PostgreSQL におけるインデックス設計として，頻出クエリに用いられる次元，すなわち address と date のカラムにだけデフォルトのインデックス B+-Tree を付与するインデキシング設計 1（計測パターン 5,7）と，全カラムに B+-Tree を付与するインデキシング設計 2（計測パターン 6,8）の 2 つを用いた．

表 4　計測パターン

Pattern name	Measurement pattern
P1	UBI-Tree (A=0, C=100) …従来の UBI-Tree と等価
P2	UBI-Tree (A=50, C=100) …クエリ履歴重視度合い：中
P3	UBI-Tree (A=100, C=100) …クエリ履歴重視度合い：強
P4	UBI-Tree (A=50, C=0) …クエリ履歴重視度合い：中, 次元種類数：無効
P5	PostgreSQL (a table, with B+-Tree indexes to dimensions in FQ)
P6	PostgreSQL (a table, with B+-Tree indexes to all dimensions)
P7	PostgreSQL (50 tables, with B+-Tree indexes to dimensions in FQ)
P8	PostgreSQL (50 tables, with B+-Tree indexes to all dimensions)

なお，UBI-Tree や PostgreSQL は Intel Core 2 Duo 3.16GHz，3.7GB Memory，CentOS 6.0 の計算機上で動作させた．また UBI-Tree の実装は C++で行い，Boost ライブラリを用いた．高速化のため，UBI-Tree においては 4.2.2.2.2 で述べた改善を行った．また UBI-Tree において E_{max}=20，E_{min}=10 とした．なお，UBI-Tree は，挿入時にメモリ上でインデキシングを行うとともに，ジャーナルファイルへの書き込みを行っているが，性能を重視するために同期書き込みは行わず，1 秒に 1 回 fsync を実行する．そこで PostgreSQL のパラメータ設定においても，fsync=off とした．また PostgreSQL の性能向上のためパラメータ設定は shared_buffers=100000 ， effective_cache_size=262144 ， random_page_count=3 ， log_min_duration_statement=0 とし，残りのパラメータ値はデフォルトのままとした．PostgreSQL のバージョンは 8.4.17 である．

4.3.3.2 実験結果

図 44 に，50 万件のデータ挿入に要する時間を計測した結果を示す．参考として，PostgreSQL においてインデックスを全く付与しないパターン（インデックスなし）でも計測を行った．クエリ履歴を考慮する新しい UBI-Tree（P1～P4）のデータ挿入性能は，従来の UBI-Tree のデータ挿入性能（図 37）とほぼ同じであった．検索性能の改善に寄与する提案手法は，データ挿入性能に悪影響を与えないと言える．また P6，P8 は他のパターンに

比べ所要時間が増加していることが分かる．これらは多数のインデックスを構築する必要があるため，性能が劣化しているものと考えられる．また P1～P4 は，インデックスなしの PostgreSQL よりもデータ挿入性能が高い．PostgreSQL においては SQL 文の複雑な構文解析を含むなど，挿入処理内容自体がやや異なるために単純に比較することはできないが，従来の UBI-Tree と同様にインデックスを付与するオーバーヘッドは十分に小さいと言えよう．

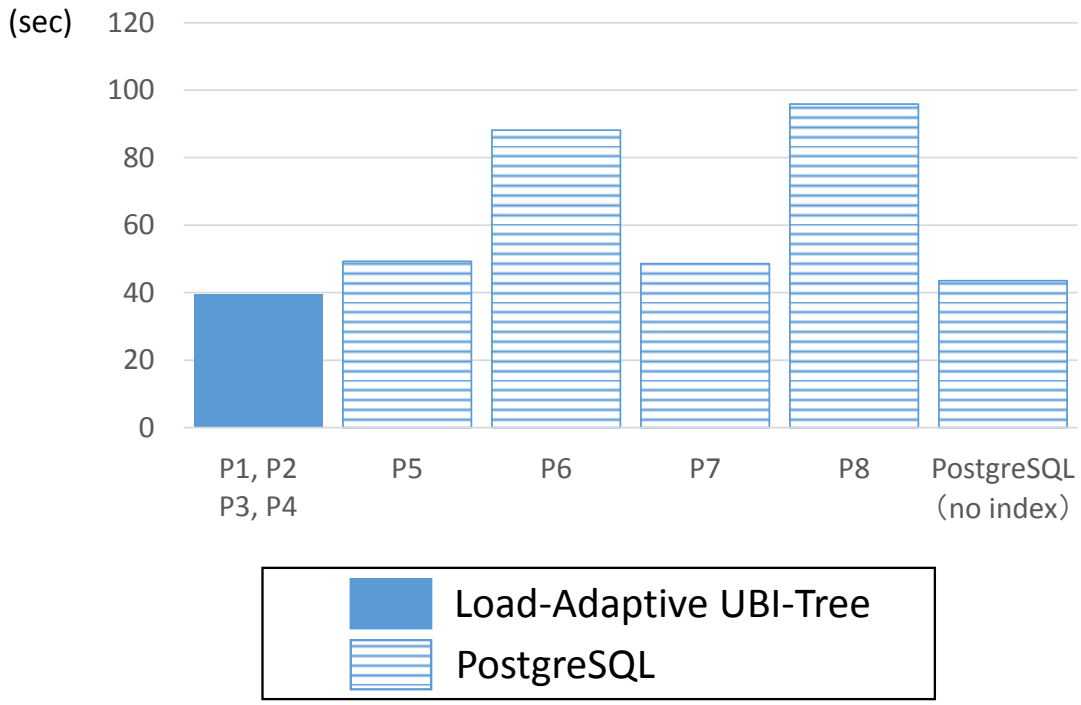
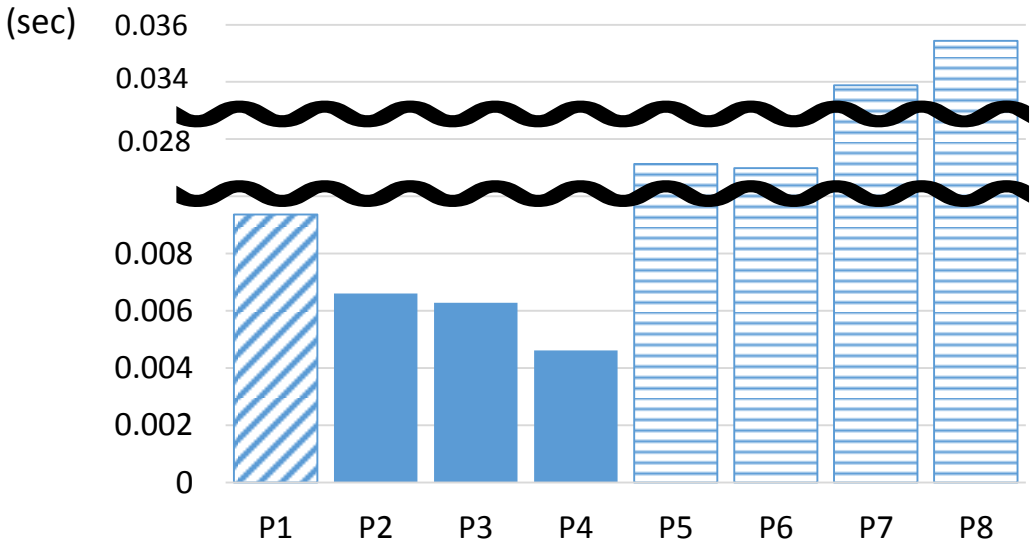
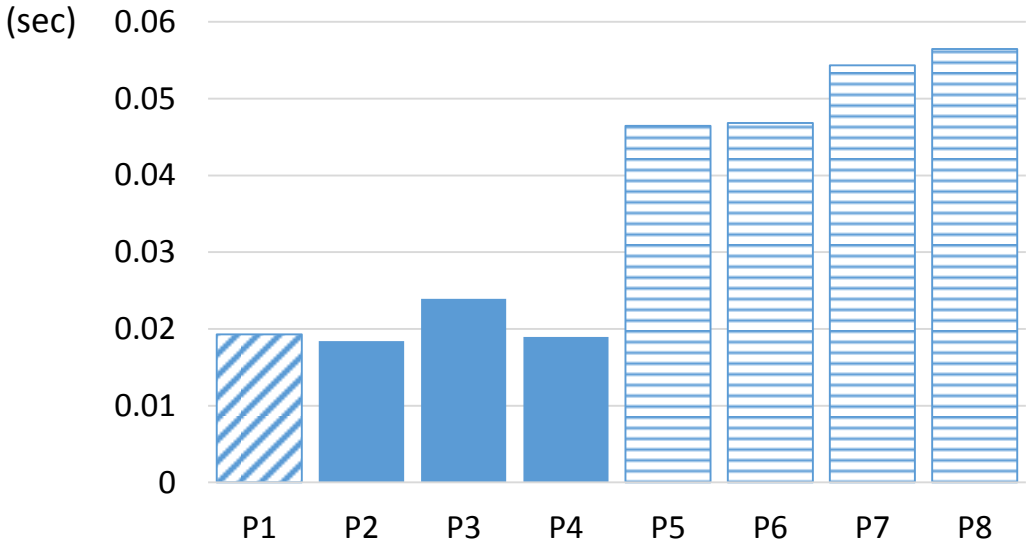


図 44　データ挿入性能

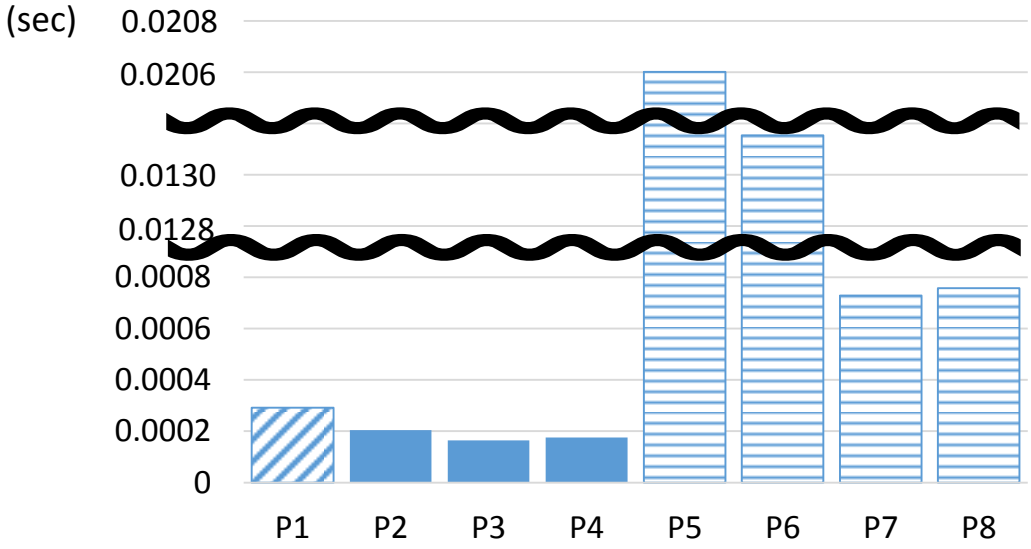
図 45 に，各種クエリでの検索に要する時間を計測した結果を示す．



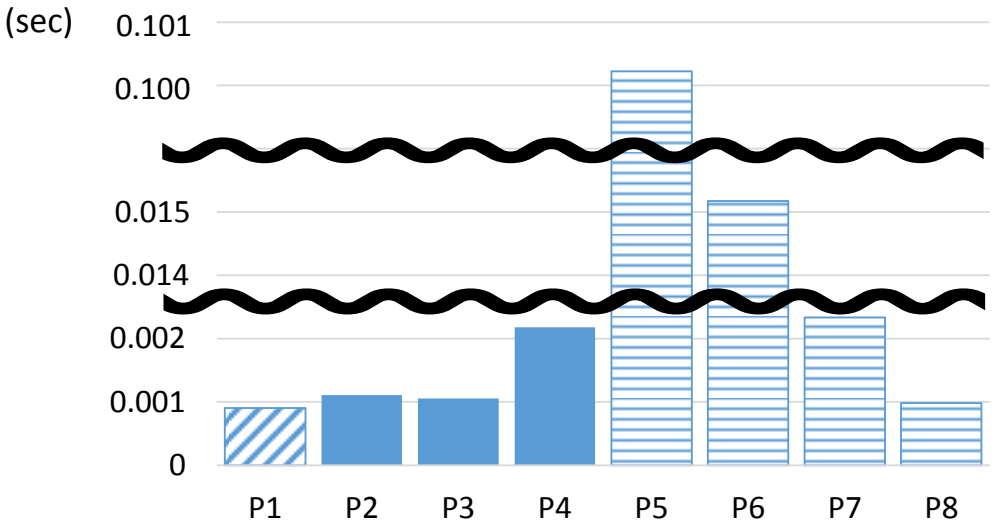
(a) FQ ＜address, date＞



(b) AQ ＜date, positionx, positiony＞



(c) AQ ＜date, uncommon dimension＞



(d) AQ ＜positionx, positiony, uncommon dimension＞

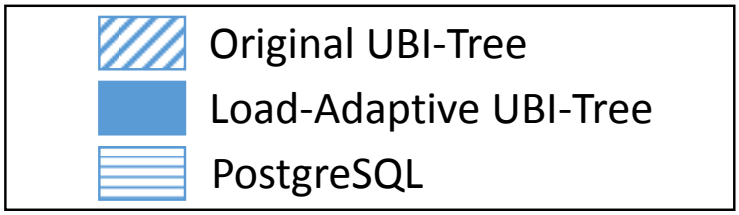


図 45　データ検索性能

図 45 (a) より，FQ＜address, date＞で検索した場合には，期待通り従来手法（P1）に比べ，提案手法（P2～P4）の検索効率が向上していることが分かる．クエリ履歴と同じクエリでの検索であるため，P2 に比べ，クエリ履歴をより重要視する P3 の検索効率の方が高い．さらに，本 FQ の検索条件に含まれる次元は全て共通次元であるため，次元種類数を無効化した P4 の検索効率が最も高くなったものと考えられる．PostgreSQL の結果（P5～P8）を見ると，1 テーブルの場合（P5,P6）に比べて 50 テーブルの場合（P7,P8）の検索効率の方が低い．これは 50 テーブルの場合，50 テーブルそれぞれを別々に検索しているためであると考えられる．

図 45 (b) より，AQ＜date, positionx, positiony＞で検索した場合には，従来手法（P1）に対し，提案手法（P3）では性能劣化することが分かる．これは，提案手法においては FQ に含まれない次元，すなわち＜positionx, positiony＞を軽視してデータ分類したために，それらの次元での絞り込みが十分に効かないためであると考えられる．しかしながら，提案手法（P2, P4）は P3 に比べ良い性能が得られている．AQ での検索のため性能が従来手法より劣化する場合であっても，クエリ履歴の統計情報だけでなく蓄積データの統計情報を併用することにより劣化度合いを低く抑えられることが分かる．なお P2 は P1 よりも若干高速化しているが，これは P2 では FQ に含まれない次元を全て軽視した分，FQ に含まれる date 次元については P1 よりも重要視することとなり，date 次元での絞り込みがよく効いたためと考えられる．

図 45 (c) より，AQ＜date, uncommon dimension＞で検索した場合には，従来手法（P1）に比べ，提案手法（P2～P4）の検索効率が向上することが分かる．提案手法ではより date 次元を重要視しており，date 次元での絞り込みがうまく効いているためであると考えられる．PostgreSQL では 1 テーブルよりも 50 テーブルの方が高速になっているが，これは当該 AQ に非共通次元が含まれるために，データ量が全体の 1/50 しか含まれていない 1 テーブルのみを検索すれば済むためであると考えられる．

図 45 (d) より，AQ＜positionx, positiony, uncommon dimension＞で検索した場合には，従来手法（P1）に比べ，提案手法（P2～P4）の検索効率が低下することが分かる．これは，当該 AQ が，提案手法において重要視される FQ に含まれる次元を全く含まないためであると考えられる．しかしながら M2M アプリケーションにおいては，通常，時間やデバイス ID など共通的なメタデータのいずれかを検索条件に含めると想定されるため，当該 AQ のような検索を効率化する重要性は低いと考えられる．また P4 は P2，P3 に比べ特に性能劣化が大きいが，これは P4 においては次元種類数を無効化するため，uncommon dimension での次元の種類による絞り込みができないためと考えられる．次元種類数を無効化した場合，図 45 (a) のように共通次元のみで検索する場合には性能を向上させることができるが，非共通次元を含む検索においては性能劣化するというトレードオフが存在していることが分かる．PostgreSQL において P6，P8 の性能が P5，P7 に比べ高いのは，AQ が含む次元

にインデックスが付与されているためであろう．

以上の実験結果をまとめると，期待通り提案手法では，AQ の性能は従来手法に比べ同程度，もしくは特に FQ に含まれていない次元を多く含む AQ において若干劣化するものの，FQ の性能は向上することが確認できた．また共通次元で検索する頻度が高い場合には，次元種類数を無効化することでさらに性能向上できることを確認することができた．提案手法は，クエリ履歴の統計情報に基づき，実際により頻繁に用いられるクエリの効率化を重視してインデキシングを行うことができ，従来に比べ，データストアの平均レイテンシやスループットを改善させることができると言える．また PostgreSQL に対し，UBI-Tree は挿入 / 検索ともに概して優れた性能を示すことを確認することができた．

4.3.4 Load-Adaptive UBI-Tree まとめ

4.3 ではまず，4.2 で述べたスキーマレス検索のためのインデキシング技術 UBI-Tree や関連技術の問題点について論じた．次に，クエリ履歴を用いた新しい UBI-Tree，Load-Adaptive UBI-Tree について提案し，実験の結果から，提案手法により負荷追従性が実現され，頻繁に用いられるクエリ（FQ）の性能が向上し，平均レイテンシやスループットの向上が可能であることを示した．

4.4 スケール性のあるシステムの検討

4.2 ではスキーマレス検索を高速化するインデキシング技術 UBI-Tree を提案し，4.3 ではさらに，負荷追従性を備えた新しい UBI-Tree，Load-Adaptive UBI-Tree について提案した．しかしながら，水平統合型プラットフォームにおける検索機能には，高速性と負荷追従性に加え，さらにスケール性が必要である．

水平統合型プラットフォームのためのデータストアとして，データ共有機構 uTupleSpace の研究開発が進められている [59]．UBI-Tree は，他のデータストアでも使用可能ではあるが，元々uTupleSpace に用いるために設計されてきたインデキシング技術である．uTupleSpace を構成する各サーバにおいて UBI-Tree を活用することにより，uTupleSpace は高速なスキーマレス検索が可能な水平統合型プラットフォームになると期待される．しかしながら，uTupleSpace は，以下で述べるようにスケール性の面で課題がある．

そこで本節では，uTupleSpace においてスケール性を実現する手法について提案する．uTupleSpace にスケール性を持たせることができれば，UBI-Tree は水平統合型プラットフ

ォームにおいてスケールする形で利用可能と言える．

以下，4.4 の構成は次の通りである．4.4.1 で uTupleSpace におけるスケール性実現の問題と，従来の負荷分散技術との比較について述べる．4.4.2 で uTupleSpace の動的スケール化手法「Dynamic-Help Method」について説明し，4.4.3 で評価実験とその結果について論じる．

4.4.1 従来技術とその問題

4.4.1.1 uTupleSpace におけるスケール性実現の問題

データ共有機構 uTupleSpace は水平統合型プラットフォームのためのデータストアであり，多様な M2M データを蓄積し，多様な M2M アプリケーション間で共有することを可能とする．より多くのセンサデータを共有するほど，M2M アプリケーションは実世界の状態やイベントをより正確に捉えることが可能となる．また，より多くのネットワーク上のアクチュエータのデータを共有するほど，M2M アプリケーションは実世界を自在に操作することが可能となる．低いコストでの迅速な M2M アプリケーション開発を可能とし，多様な M2M アプリケーションの出現を促す水平統合型プラットフォームの意義から考えると，uTupleSpace のスケール性を実現することは非常に重要である．

uTupleSpace は，並列コンピューティングにおいて用いられる通信モデルである tuple space をベースにしている [60]．tuple space は，共有メモリ空間に対して actual（あるいは entry）と formal（あるいは template）と呼ばれる 2 種類のデータ（タプル）を読み書きすることによって，多対多の通信を行う通信モデル，あるいは当該共有メモリ空間を指す．これらを一般的な DBMS の概念に置き換えれば，actual はデータそのもの，formal は検索条件に対応し，actual を tuple space 内に次々と蓄積していくことが M2M データの履歴蓄積に相当する．その際，これら 2 種のタプルの登録順序によらずマッチングプロセスが行われ結果が通知される点が特徴である．すなわち M2M における応用の観点で捉えるならば，M2M データの即時配信と蓄積検索をモデル上でシームレスに統合できる本質的利点をもつと言える．uTupleSpace は，この tupe space に後述する拡張を施し，M2M アプリケーション構築に対する有用性を高めたものである．

一方，3.1 で述べたように，分散ハッシュテーブル（DHT）は近年注目されているスケール性のあるデータストアである．DHT は多くのピアに分散した形で巨大なハッシュテーブルを管理するシステムである．DHT は代表的なスケールアウト技術として近年多くの研究がなされている．DHT は uTupleSpace のスケール性を実現する技術として強いポテンシャルを持っていると考えられる．

しかしながら，従来の DHT はピア間で必ずしも十分に負荷を分散することができない．

各ピアは全ハッシュ空間の一部の区間を管理し，当該ハッシュ区間内のハッシュ値をもつデータにより発生する負荷を担う．DHT は最初に，どのピアがどのハッシュ区間を管理するかを決定する．DHT における従来の負荷分散手法のほとんどは，その後，各区間の負荷に応じて，各区間の大きさを調整する．しかしながら，それら手法は特定のハッシュ値に集中した負荷に対応することができない．いくら区間の大きさを 1 つのハッシュ値にまで小さくしたとしても，当該区間を管理するピアが当該ハッシュ値の負荷を担いきれなくなるためである．さらに，各ハッシュ値に蓄積されたデータの量が非常に大きい場合，負荷分散，すなわち各区間の大きさの調整に伴うピア間でのデータ移動プロセスの負荷が非常に大きくなってしまう．

DHT は分散キーの値によってデータを分散させる．そのため，負荷の分散の仕方は分散キー次第となる．各データは 1 つ以上のキー-バリューペア（key-value pair）を含み，分散キーは各データが含むキーの 1 つである．分散キーの値はハッシュ関数の引数として使われるため，全ての挿入／検索プロセスで必要となる．それゆえ，DHT を用いた uTupleSpace を設計する際には，センサデータに含まれ，アプリケーションが指定する検索条件に含まれるキーを分散キーに選ぶ必要がある．例えば，計測時刻やセンサ種別はセンサデータや検索条件に通常含められると考えられるため，分散キーとなり得る．しかしながら，同じ計測時刻，同じセンサ種別のセンサデータは大量に存在し得るため，負荷の集中が発生し得る．2 つ以上のキーから成る分散キーを用いれば，1 つのハッシュ値に関係づけられるデータの量を減らすことができ，問題をある程度緩和することはできる．しかしながら，問題を完全に解決することはできず，さらに挿入／検索プロセスにおいて値を指定しなければならないキーの数が増えるため，ユーザビリティを損なうことに繋がってしまう．従来の DHT においては，1 つのハッシュ値に関係づけられるデータの量はそれほど大きくないという前提があるように思われる．しかしながら，水平統合型プラットフォームのデータストアである uTupleSpace においては，そのような前提を置くことができない．

4.4 で述べる uTupleSpace のための新しい負荷分散手法は，1 つのハッシュ値に関係づけられる負荷を 1 つ以上のピアで担うことを可能とする．これにより，1 つのハッシュ値に関係づけられる巨大な負荷を複数のピアに分散させ，処理することが可能となる．さらに，提案手法により，負荷分散に伴うデータ移動プロセスの負荷が小さくなる．これにより，データ移動プロセスの負荷による挿入／検索プロセスの滞留を防ぐことができる．マイクロベンチマークにより提案手法の性能を測定した結果，低いオーバーヘッドで十分な負荷分散が実現できることが確認された．またシミュレーションにより提案手法のフィージビリティを評価した結果，提案手法により uTupleSpace は増加する負荷に対して単純なオペレーションルールで安定的かつ低コストに対応可能となることが確認された．

4.4.1.2 その他の従来技術とその問題

tuple space は 1 つの共有メモリ空間に通信が集中するモデルであるため，スケール性を持たせることは簡単ではない．tuple space のスケール性を実現するため，多くの研究がなされてきた．

TinyLIME [61]と TeenyLIME [62]は，ローカルな無線通信接続を用いて利用可能な端末間でアドホックな tuple space を形成する．W4TupleSpace [63]は，タプルを W4（Who, What, Where and When）と呼ばれる統一形式に変換し，地理的，あるいはトピックによって分割した tuple space の 1 つに格納する．Agimone [64]は 2 つのシステム，すなわち，無線センサネットワーク（WSNs; Wireless Sensor Networks）のための Agilla [65]と，IP ネットワークのための Limone [66]を統合する技術である．Agimone は各 WSN において tuple space を形成し，事前に設定した tuple space 間でのみ通信することを可能とする．これらの手法は地理的範囲やネットワークの接続性により予め定められた単位で tuple space を分割することにより，1 つの tuple space が大きくなり過ぎるのを防ぎ，大量のデータや通信を収容可能とするものである．しかしながら，様々なアプリケーションによりセンサデータを共有する場合，アプリケーションによって必要とするセンサデータの範囲は異なるため，分割の単位を予め定めることは困難である．予め定められた範囲で分割することは，柔軟性を損ない，アプリケーション開発者を制限することに繋がってしまう．

De et al. [67]は，tuple space を分割する代わりにマッチングプロセスを高速化することにより，スケール性を実現しようとするものである．しかしながら，検索時間はタプルの数に比例し，十分なスケール性を確保できたとは言い難い．

DTuples [68]と BISSA [69]は各タプルに subject キーまたは message キーを含ませ，それを分散キーとして用い，DHT により巨大な tuple space を構築する．これらのシステムの負荷分散手法は，用いる DHT に依存する．

一方，DHT の負荷分散についても多数の研究が行われてきた．Chord [29]は，ピアの多様性を許容するため，バーチャルサーバの概念を導入している．DHT に参加するピアは異なる数のバーチャルサーバを動作させることにより，各ピアの異なる性能を最大限に活かすことができる．DHT における負荷分散のための研究のほとんどは，どのようにバーチャルサーバを管理するかに焦点を当てている [70] [71] [72].

A. Rao et al. [70]は，過負荷となったピアで動作しているバーチャルサーバをより負荷の小さなピアへ再配置する手法を提案している．バーチャルサーバの再配置に伴うデータ移動プロセスは計算量的コストが高いため，技術的問題は負荷の均等性を損なうことなくいかにそのコストを抑えるかということになる．この問題を解決するため，S. Surana et al. [71]や C. Chen et al. [72]では，中央管理サーバがピア間でのバーチャルサーバの再配置を管理する，集中的負荷均等化アルゴリズムが導入された．さらに，L. Yang et al. [73]は，ピアが 1 つのバーチャルサーバしか動作させていないにも関わらず当該ピアが過負荷になっ

た場合に，バーチャルサーバを分割する手法を提案した．

しかしながら，これら従来の手法は，1 つのハッシュ値に関係づけられる負荷によりピアが過負荷になってしまうケースに対処することができない．これらの手法では，1 つのハッシュ値に関係づけられるデータは，ただ 1 つのピアにより収容されるためである．特に uTupleSpace の場合には，前述の通り 1 つのハッシュ値に関係づけられるデータ量が大きくなりがちであるため，この問題は重大な問題となる．一方，提案手法は，データを冗長に複数のピアで収容することを許容することで，1 つのハッシュ値がホットスポットとなり，当該ハッシュ値に関係づけられるデータが大量になった場合にも対応可能とする．

これらの手法は中央集中型のアルゴリズムを採用し，負荷分散のための中央管理サーバを導入したため，当該サーバが性能ボトルネック，あるいは Single Point of Failure となり得る．それゆえ，Hsiao [74]は負荷分散のための分散型アルゴリズムを提案した．提案手法は中央管理サーバを用いるが，分散型アルゴリズムを採用し，中央管理サーバを不要とすることも可能であろう．

4.4.2 提案手法「Dyamic-Help Method」を用いた uTupleSpace

本節では，提案手法，すなわち，動的に負荷を分散し，uTupleSpace のスケール性を実現する Dynamic-Help Method について説明する．本手法は，uTupleSpace 内に存在するサーバ間でできる限り負荷を分散し，負荷が当該サーバ群の容量を超えた場合に初めて，uTupleSpace に新しいサーバを追加するようシステムオペレータに要求する．このアプローチは，効率的に既存リソースを活用し，設備投資を最低限に抑えることを可能とする．

負荷は計算負荷と記憶負荷に分類することができる．前者は CPU 負荷や I/O 負荷を含む．後者はハードディスクに蓄積されたデータの量，つまりハードディスク使用量を指す．以降，本節では，まず uTupleSpace モデルの詳細と，提案手法の基本的アイデアについて述べる．その後，システムの概要を示し，計算負荷，記憶負荷それぞれに対する動的負荷分散の例について説明する．加えて，システムへの新しいサーバの追加方法についても説明する．また，付録 B にて，更なる性能向上のための手法について議論する．

4.4.2.1 uTupleSpace モデル

uTupleSpace モデルでは，タプルにメタデータフィールドが導入されている．uTupleSpace モデルは，tuple space モデルを基本としつつ，メタデータに対する範囲条件を指定したマッチング機能の拡張，および，双方向通信のためのメタデータに対する下り方向（M2M アプリケーションからデバイスへの送信）マッチング規則の拡張を施したものである．これによりセンサデータの読み出しにおいて多くの M2M アプリケーションが必要

とする範囲検索を可能とし，さらに操作対象とするアクチュエータデバイスをコマンド送信時に柔軟に範囲指定することを可能としている．

uTupleSpace において用いられるタプルの形式を uTuple と呼び，uTuple はメタデータとデータから構成される．表 5 に uTuple のデータ形式を示す．メタデータには address（デバイス ID)，データが生成された時刻，その際デバイスが存在した位置が含まれる．これらのフィールドには値として範囲条件を設定することができる．またメタデータにはデバイスやデータの種別が含まれる．これらのフィールドは DHT の分散キーとして用いるため，これらのフィールドに範囲条件を設定することはできない．

表 5　uTuple のデータ形式

Metadata	Address	Identifier of sensor / actuator
	Time	Time when data were stored by sensor / actuator
	Position	Position (latitude and longitude) of sensor / actuator
	Subject	Type of sensor / actuator
	Type	Type of data
Data	(user-defined)	

uTupleSpace モデルは 2 種の通信，すなわち選択的読み出しを行うイベント通信と，選択的書き込みを行うコマンド通信をサポートする．uTupleSpace モデルを図 46 に示す．イベント通信が tuple space モデルに元々存在する通信であるのに対し，コマンド通信は uTupleSpace における拡張である．各通信に用いられる uTuple ペアのメタデータは異なる．すなわち，イベント通信において，書き手は値を設定し，読み手は範囲条件を設定する．コマンド通信においては，それが逆になる．

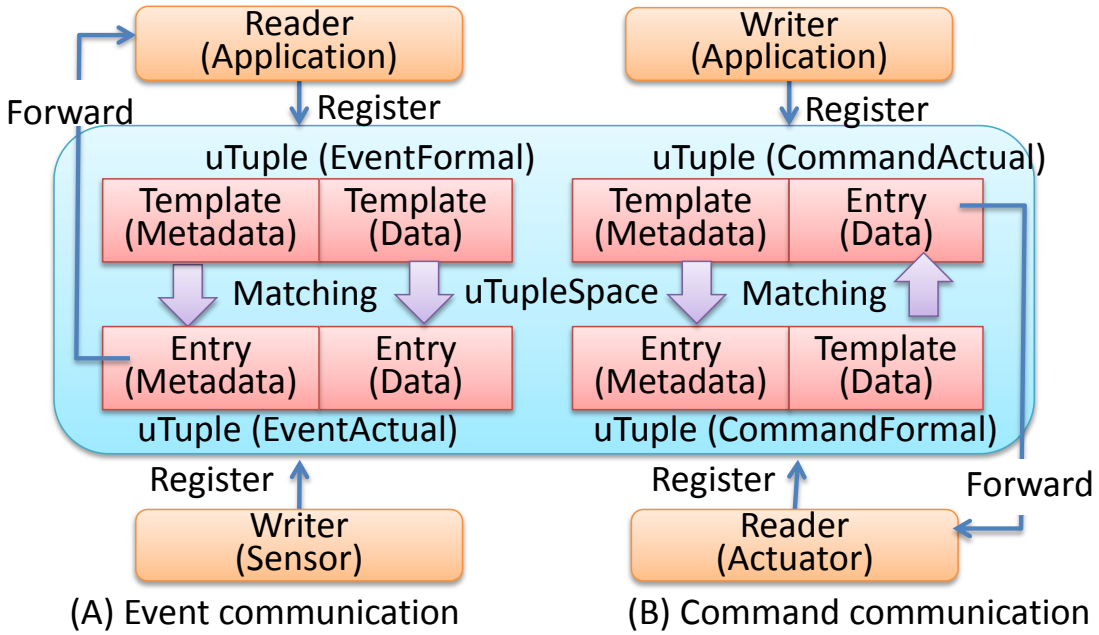


図 46　uTupleSpace モデル

イベント通信においては，書き手（センサプログラム）は EventActual uTuple（EA）を uTupleSpace に登録する．EA は書き手自身の情報を含むメタデータと，データから構成される．読み手（アプリケーションプログラム）は EventFormal uTuple（EF）を登録する．EF は求めるデータや書き手とマッチするための検索条件を含むメタデータとデータから構成される．コマンド通信においては，読み手（アクチュエータプログラム）は CommandFormal uTuple（CF）を登録する．CF は読み手自身の情報を含むメタデータと，求めるデータとマッチするための検索条件を含むデータから構成される．書き手（アプリケーションプログラム）は CommandActual uTuple（CA）を登録する．CA はデータと，求める読み手，すなわち操作対象とするアクチュエータデバイスとマッチするための検索条件を含むメタデータから構成される．これらの通信により，空間範囲や時間範囲などを指定した，選択的読み出しおよび選択的書き込みが達成される．このように uTuple を用いることで，読み出しや書き込みの柔軟なスコープ設定が可能である．

uTuple が uTupleSpace に登録された際，当該 uTuple はカウンターパートとなる uTuple 群と照合される（マッチングプロセス）．もし当該 uTuple が EA の場合，カウンターパートは EF である．CA の場合，カウンターパートは CF である．また同様に，EF に対しては EA，CF に対しては CA がそれぞれカウンターパートとなる．マッチングプロセスにおいて EA や CA が EF や CF とマッチした場合（つまり，CA や EF に含まれる検索条件を CF や EA が満たした場合），EA や CA はそれぞれ EF や CF を登録した読み手に送信され

る．次に，登録された当該 uTuple はハードディスクに書き込まれ，蓄積される（永続化プロセス）．各 uTuple には有効期間を設定することができ，有効期間を過ぎた uTuple は，定期的に実行されるガベージコレクション処理により削除される．有効期間を 0 に設定された uTuple が登録された場合，マッチングプロセスのみ実行され，永続化プロセスはスキップされる．

4.4.2.2 Dynamic-Help Method の基本的アイデア

サーバ A の計算負荷が過剰になった場合，次のように提案手法を用いてその負荷の一部を負荷の軽いサーバ B に担わせる．

センサプログラムは，それまでサーバ A に登録すべきであった EA を，それ以降は A，B のどちらかにだけ登録することで，A の負荷の一部を B に移す．しかしながら，アプリケーションプログラムは，それまで A に登録すべきであった EF を，それ以降は A と B 両方に登録する．これは，もし A か B のどちらかにだけ EF を登録してしまうと，センサプログラムと十分に通信すること，つまり，EF とマッチする EA を漏れなく読み出すことができなくなってしまう．これでは，プログラム開発者が意図した動作を実現することができない．

EF を 1 つのサーバにのみ登録し，EA を両方のサーバに登録しても，通信の完全性は実現できる．しかしながら，効率的な負荷分散を実現するには，EF を両方のサーバに登録し，EA を 1 つのサーバにのみ登録すべきである．なぜなら，ほとんどの M2M 環境において，センサデータは通常大量となる一方，検索クエリの数はずっと少ない傾向にあるためである．つまり，センサプログラムにより登録される EA の数は，アプリケーションプログラムにより登録される EF の数よりずっと大きい．M2M 環境におけるこの特徴により，提案手法は高い負荷分散効率を達成することができ，またタプルの複製によるオーバーヘッドを小さくすることができる．

提案手法は，同様の考察を CF と CA についても適用するが，完全に同じように適用することはできない．なぜならば，EA の蓄積数と EF の蓄積数の関係は，EA の登録頻度と EF の登録頻度の関係と同様，すなわち，EA 蓄積数 ≫ EF 蓄積数 と EA 登録頻度 ≫ EF 登録頻度 という関係であるのに対し，CF と CA では異なる傾向にあるためである．すなわち，CF 蓄積数 ≫ CA 蓄積数 である一方，CF 登録頻度 ≪ CA 登録頻度 という関係がある．この理由は次の通りである．

第一に，アクチュエータデバイスは 1 回以上操作される．これはつまり，アクチュエータデバイスは 2 回以上の CA を受信することを意味する．また，CF の数はアクチュエータデバイスの数に比例する．それゆえ，CF 登録頻度 ≪ CA 登録頻度 が成り立つ．第二に，アクチュエータデバイスの操作は実世界に影響を及ぼすため，ほとんどの M2M アプリケーションは，操作権限のある特定のアクチュエータデバイスのみを，操作コマンドを発行した

ときにだけ操作する．このため，ほとんどの CA は Address フィールドに特定のデバイス ID を設定され，また有効期間は 0 を設定される（以降，この単純な CA を sCA（simple CA）と呼ぶこととする）．sCA の登録において永続化プロセスはスキップされる．それゆえ，CF 蓄積数 ≫ CA 蓄積数 が成り立つ．CF と CA が持つこの特徴のため，EA / EF の場合と同じように，CF を A か B のどちらかにだけ登録し，CA を A，B 両方に登録するという単純な手法では，CA の大きな負荷が A，B 両方にかかってしまうという問題が発生する．

そのため，sCA の処理に特化したサーバを提案手法では新たに導入する．このサーバを sCA サーバと呼ぶ．CF が登録された場合，CF を複製し，1 つは EA と同様に A または B のどちからにだけ登録する．もう 1 つは sCA とマッチングプロセスを行うため sCA サーバに登録する．

CA が登録された場合，まず当該 CA が sCA かどうかを検査する．もし sCA でないのであれば，EF と同様に A と B 両方に登録する．もし sCA の場合には，sCA サーバに登録し，そこに蓄積された CF とマッチングプロセスを実行する．sCA サーバでは，マッチングプロセスは uTuple の 3 つのフィールド，すなわち，Subject / Type / Address において，両者の値が完全一致するか否かを検査するだけで良い．さらに，sCA の有効期間は 0 であるから，永続化プロセスをスキップすることができる．このように，sCA サーバでは単純な処理のみを行なえば良いため，sCA サーバは高いスループットを実現することができる．sCA サーバが負荷分散しなければならない場合も，すなわち，3 つのフィールドを連結させたものを分散キーとして用いた従来の DHT の負荷分散手法が有効に機能するであろう．

サーバ A の記憶負荷が過剰になった場合には，その負荷の一部を負荷の軽いサーバ B に担わせる．この場合，これ以上 A に uTuple を登録することができない．しかしながら，もし全ての uTuple を B だけに登録すると，通信の完全性が損なわれてしまう（それまで A に登録されていた uTuple とのマッチングプロセスが実行されない）．このため，アプリケーションプログラムは EF / CA を B だけでなく A にも登録し，A への登録においては uTuple の有効期間を 0 に設定することで永続化プロセスをスキップさせる．また，それまでに A に蓄積されていた EF / CA は B にコピーする．これにより，それまで A に登録されていた EF / CA と新たに登録される EA / CF とのマッチングプロセスが確実に実行されるようになる．蓄積された EF の数は EA を継続的に検索するアプリケーションの数に等しく，蓄積された CA の数は sCA でない CA の数に等しいため，蓄積された EF / CA の数はともに比較的少なく，EF / CA のコピーに伴うオーバーヘッドも小さい．

4.4.2.3 システム概要

提案手法を適用するシステムの概要を図 47 に示す．

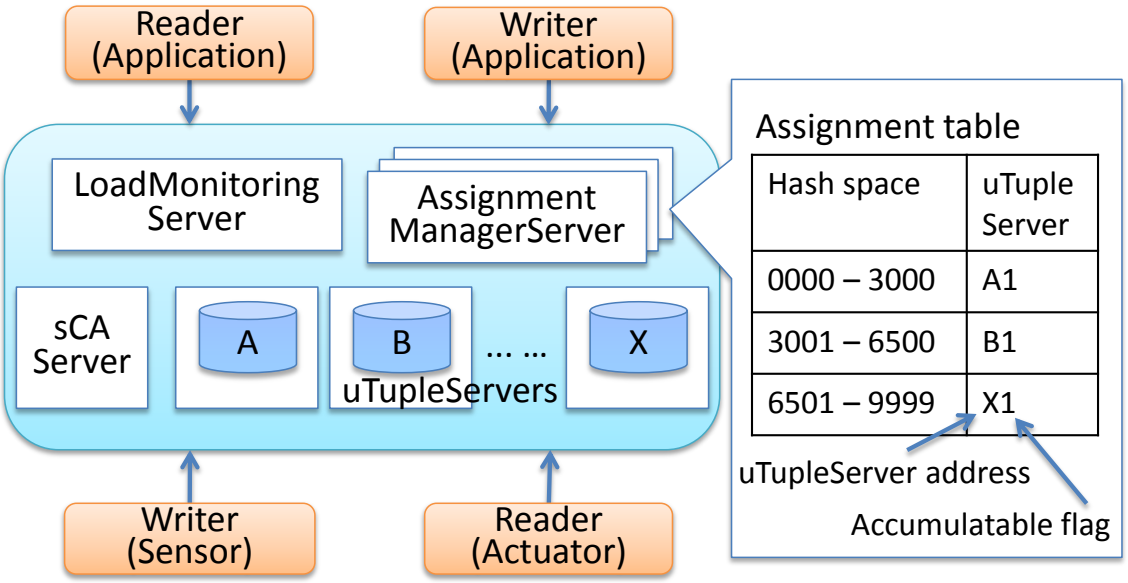


図 47　システム概要

uTuple サーバ（uTupleServer）はマッチングプロセスの実行と uTuple の蓄積を行う．負荷監視サーバ（Load Monitoring Server）は各 uTuple サーバの負荷を監視し，必要に応じてそのバランスをとる．分担管理サーバ（Assignment Manager Server）は，どの uTuple をどの uTuple サーバに登録すべきかを示す分担表（assignment table）を管理する．分担表にはハッシュ空間を分割した各ハッシュ区間に対し，当該ハッシュ区間を担当する uTuple サーバのアドレスが記述されている．またアドレスには蓄積可能フラグ（accumulatable flag）が付与される．蓄積可能フラグについては後述する．分担管理サーバは複数台設置し，DHT によって分担表を分散管理する．DHT は分担管理サーバ上で動作するピアにより形成される．バーチャルサーバを用いて，uTuple サーバと同数のピアを起動する．ここで用いる DHT は，通常の DHT とクエリルーティングなどの基本的なアルゴリズムは同じであるが，次の点でやや異なる．各ピアは自身が担当するハッシュ区間の範囲情報と，当該ハッシュ区間を担当する uTuple サーバのアドレスと蓄積可能フラグのリストとを保持する．ピアがクエリを受け取った場合には，データではなく当該リストを返す．新しい uTuple サーバが増設された場合には，それに対応する新しいピアが起動され，DHT に加える．各ピアが保持する情報量は非常に小さく，メモリ上に保持可能であり，クエリ処理をメモリアクセスのみで実行することができるため，DHT は高いスループットを示す．CA サーバは sCA と CF 間のマッチングプロセスと，CF の蓄積を行う．各サーバはイーサネットで接続されていることを想定している．提案手法はこれらサーバ群により実現される．

提案手法の利点は,同じ分散キーをもつuTupleにより生じた負荷を分散できる点にある.提案手法は主たる負荷，すなわち，マッチングプロセスと永続化プロセスによる負荷を，DHT のピアから切り離し，分散した形で uTuple サーバに担わせる．システム規模が十分に小さい場合には 1 つの分担管理サーバだけで十分であり，そのような場合には分担表をDHT で分散管理する必要はない.

クライアントプログラム（uTuple 登録元のセンサ／アクチュエータ／アプリケーションプログラム）は uTuple 内の特定の 2 つの値，すなわち，Subject キーの値と Type キーの値を連結したものをハッシュ関数にかけ，得られたハッシュ値を分散キーとして用いる．クライアントプログラムは当該分散キーを用いてDHTに問合せ,分担表を用いて当該uTupleを登録すべき uTuple サーバのアドレス群を取得する．次に，クライアントプログラムはuTupleをuTupleサーバに登録する.しかしながら,CFを登録する際には,上述したuTupleサーバへの登録に加え，クライアントプログラムは sCA サーバへも登録を行う．sCA を登録する際には，クライアントプログラムは uTuple サーバへの登録を行わず，sCA サーバにのみ行う.

uTuple サーバへの登録においては，uTuple をどの uTuple サーバに登録するかを決定するため，uTuple 内には Subject キー（デバイス種別）と Type キー（データ種別）の値を含めることが要求される.しかしながら,M2M 環境におけるアプリケーションプログラムは,常にこれらのキーの値を認識しながら通信を行うと考えられるため，本要求は実質的に制約にはならない.

4.4.2.4 計算負荷の動的分散例

1.uTuple サーバ A の計算負荷が閾値 T_p を超えていることを負荷監視サーバが検知する.

2．負荷監視サーバは，uTuple サーバ群の中から，その時点で負荷が最も軽い uTuple サーバ B を，uTuple サーバ A の負荷を分担するサーバ（ヘルパーサーバ）として選出する.

3．負荷監視サーバは，それまでに uTuple サーバ A に蓄積した EF と CA を uTuple サーバ B にコピーするよう uTuple サーバ A に指示する.uTuple サーバ A はコピーを実行する.

4．負荷監視サーバは，分担表の中で uTuple サーバ A のアドレスが含まれる行に，uTuple サーバ B のアドレスを追加するよう分担管理サーバに指示する．分担管理サーバは追加を実行する．追加後の分担表を図 48 に示す．図中の「1」は，蓄積可能フラグは ON に設定されていることを示している.

5．クライアントプログラムは，uTuple サーバ A が担当しているハッシュ区間に関係づけられる分散キーを用いて DHT に問合せた際，分担管理サーバから uTuple サーバ A,

B 両方のアドレスを取得する．アプリケーションプログラムが EF または CA を登録する際には，uTuple サーバ A，B 両方に登録する．デバイスプログラムが EA または CF を登録する際には，uTuple サーバ A，B のどちらかをランダムに選び，選んだ方へ EA または CF を登録する．

Hash space	uTupleServer
0000 – 3000	A1
3001 – 6500	B1
6501 – 9999	X1

Hash space	uTupleServer
0000 – 3000	A1, **B1**
3001 – 6500	B1
6501 – 9999	X1

図 48　計算負荷の動的分散による分担表更新の例

4.4.2.5 記憶負荷の動的分散例

1．uTuple サーバ A の記憶負荷が閾値 T_m を超えていることを負荷監視サーバが検知する．

2．負荷監視サーバは，uTuple サーバ群の中から，その時点で負荷が最も軽い uTuple サーバ B を，uTuple サーバ A の負荷を分担するヘルパーサーバとして選出する．

3．負荷監視サーバは，それまでに uTuple サーバ A に蓄積した EF と CA を uTuple サーバ B にコピーするよう uTuple サーバ A に指示する．uTuple サーバ A はコピーを実行する．

4．負荷監視サーバは，分担表の中で uTuple サーバ A の蓄積可能フラグを全て OFF に設定するとともに，uTuple サーバ A のアドレスが含まれる行に，uTuple サーバ B のアドレスを蓄積可能フラグを ON で追加するよう分担管理サーバに指示する．分担管理サーバは設定・追加を実行する．設定・追加後の分担表を図 49 に示す．

5．クライアントプログラムは，uTuple サーバ A が担当しているハッシュ区間に関係づけられる分散キーを用いて DHT に問合せた際，分担管理サーバから蓄積可能フラグが OFF になった uTuple サーバ A のアドレスと，蓄積可能フラグが ON の uTuple サーバ B のアドレスを取得する．アプリケーションプログラムが EF または CA を登録する際には，uTuple サーバ A へ有効期間 0 で登録するとともに，uTuple サーバ B へ元々の有効期間で登録する．デバイスプログラムが EA または CF を登録する際には，蓄積可能フラグが ON である uTuple サーバ B にのみ登録する．

Hash space	uTupleServer
0000 – 3000	A1
3001 – 6500	B1
6501 – 9999	X1

Hash space	uTupleServer
0000 – 3000	A**0**, **B1**
3001 – 6500	B1
6501 – 9999	X1

図 49　記憶負荷の動的分散による分担表更新の例

4.4.2.6 uTupleSpace への新しい uTuple サーバ増設

負荷が増加し，既設の uTuple サーバ群の容量の限界にまで達した場合，新しい uTuple サーバを uTupleSpace に増設しなければならない. 新しい uTuple サーバは, 既設の uTuple サーバからハッシュ区間の一部を引き継ぐことで uTupleSpace に加わる. この手法は DHT に新たなピアを追加する手法と同様である．しかしながら，既存の uTuple サーバは，自身が担当するハッシュ区間の一部を新しい uTuple サーバに引き継ぐ際，当該一部の区間に関係づけられる全ての uTuple を新しい uTuple サーバに移すわけではないという点で異なる. 特に EA や CF については数が多く，その移し替えに伴うコストが極めて高いからである. 本手法の例を以下に示す.

1．新しい uTuple サーバ Z は，負荷監視サーバにサーバ追加要求を送信する.

2．負荷監視サーバは，uTuple サーバ群の中から，その時点で負荷が最も重い uTuple サーバ A を選出する. 負荷監視サーバは，分担管理サーバに指示し，uTuple サーバ A により管理されるハッシュ区間を分割させる. 具体的には，まず uTuple サーバ A に，蓄積した uTuple の中からランダムに 1 つ選ばせ，負荷監視サーバにそのハッシュ値を通知させる. 次に，負荷監視サーバは分担監視サーバに，uTuple サーバ Z に対応する新しいピアを起動し，当該ハッシュ値と，uTuple サーバ A と Z のアドレスをパラメータとして DHT に加えるよう指示する. 分担管理サーバは新しいピアを起動し，DHT に追加する. 追加時，当該ハッシュ値を含むハッシュ区間を担当するピアは，当該ハッシュ区間を半分に分割し，一方を自身で担当するとともに，他方を新しいピアに担当させ，また当該ハッシュ区間を担当する uTuple サーバのリストを新しいピアにコピーする.

3．負荷監視サーバは，それまでに uTuple サーバ A に蓄積した，新しいピアが担当するハッシュ区間に関係づけられる分散キーをもつ EF と CA を uTuple サーバ Z にコピーするよう uTuple サーバ A に指示する. uTuple サーバ A はコピーを実行する.

4．負荷監視サーバは，分担表の中で uTuple サーバ A の蓄積可能フラグを全て OFF に設定するとともに，uTuple サーバ A のアドレスが含まれる行に，uTuple サーバ B のア

ドレスを蓄積可能フラグ ON で追加するよう分担管理サーバに指示する．分担管理サーバは設定・追加を実行する．設定・追加後の分担表を図 50 に示す．

５．その後の動作は記憶負荷の動的分散のステップ５と同じである．

Hash space	uTupleServer
0000 – 3000	A1
3001 – 6500	B1
6501 – 9999	X1

Hash space	uTupleServer
0000 – **1500**	A1
1501 – **3000**	**A0**, **Z1**
3001 – 6500	B1
6501 – 9999	X1

図 50　新しい uTuple サーバの増設による分担表更新の例

4.4.3 評価

提案手法の性能とフィージビリティについて評価を行った．結果を以下に記述する．

4.4.3.1 性能評価

スループットと記憶容量の観点から性能評価を行うため，まず uTuple サーバ上でマイクロベンチマークを実行した．次に，マイクロベンチマークの結果から，想定アプリケーションの負荷に対するシステム全体の性能を見積もった．

提案手法により uTuple サーバの数に比例して uTupleSpace の性能が向上すれば理想的であるが，実際には性能は負荷の分散状況に依存する．もし負荷がハッシュ空間上で均一に分散するのであれば，つまり各ハッシュ値に関係づけられるデータ量およびアクセス量が均等であれば，動的負荷分散をせずとも DHT によって uTuple サーバ間で負荷は均等に分散され，uTupleSpace は理想的な性能を示すであろう（以降，このような状況を「最良の状況」と呼ぶ）．もし負荷が不均一に分散するのであれば，EF や CA の複製が必要となり，これにより計算量および記憶量のオーバーヘッドが発生する．最悪の状況は，負荷が 1 つのハッシュ値に集中し，このオーバーヘッドが最大に達する状況である．従来の手法では，この最悪の状況において負荷を分散させることができない．それゆえ，最良の状況と最悪の状況それぞれにおいて，uTuple サーバの数の変化に対し，スループットと蓄積可能な uTuple

の数がどのように変化するかを評価した.

4.4.3.1.1 マイクロベンチマーク結果

uTuple サーバは Xserve（Quad-Core Intel Xeon 2.8GHz，2GB メモリ）上で動作させた．uTuple サーバには EF，EA，CF を事前に登録しておき，その状態で EA，EF，CA の uTuple を登録する際の性能をそれぞれ測定した.

事前に登録しておく uTuple の数を変化させた場合の，uTuple の登録に要する実行時間の変化を図 51 に示す．このとき各登録においては，uTuple はマッチングプロセスで他の 1 つの uTuple とマッチするようにした．また，蓄積している EA と EF に消費しているメモリサイズを計測した結果を図 52 に示す．各 EA は，後述するアプリケーション A および B で想定される少量のデータを Body フィールドに含ませた.

なお，uTuple サーバは PostgreSQL を用いて実装した．各種 uTuple はそれぞれ別のテーブルに格納し，メタデータに対応するカラムにはデフォルトのインデックス（B+-Tree）を付与し，マッチングプロセスを高速化した.

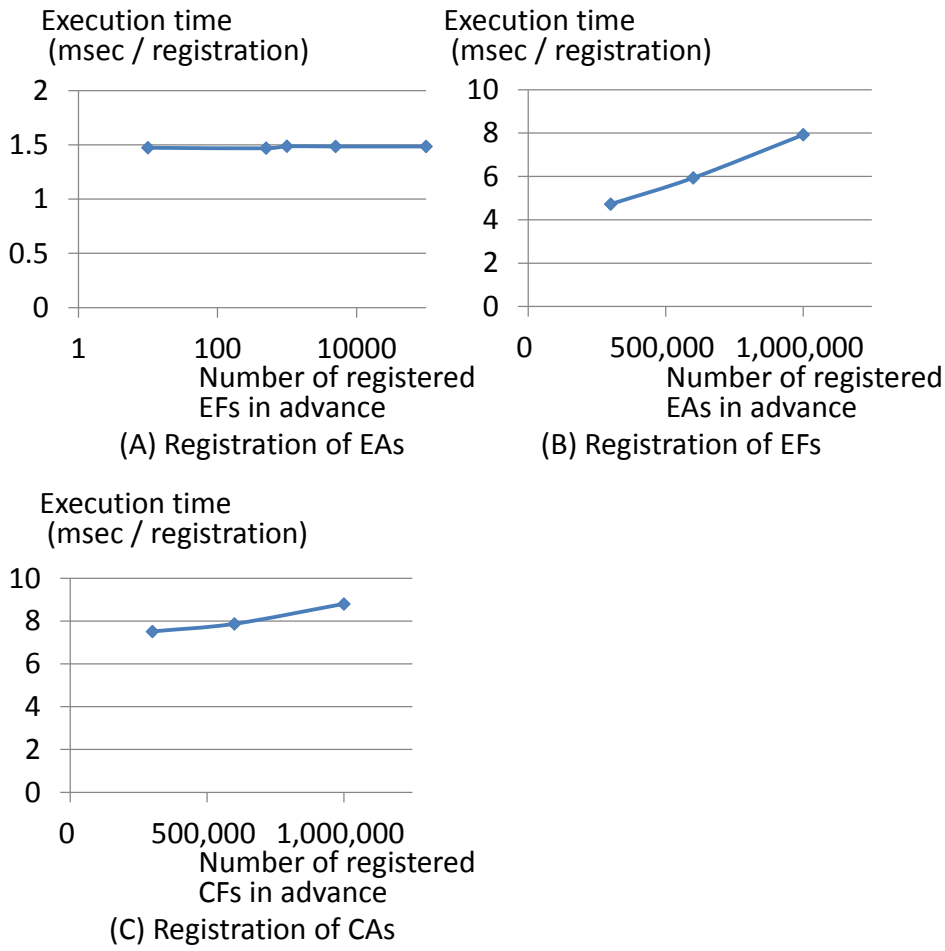


図 51　uTuple の登録に要した実行時間

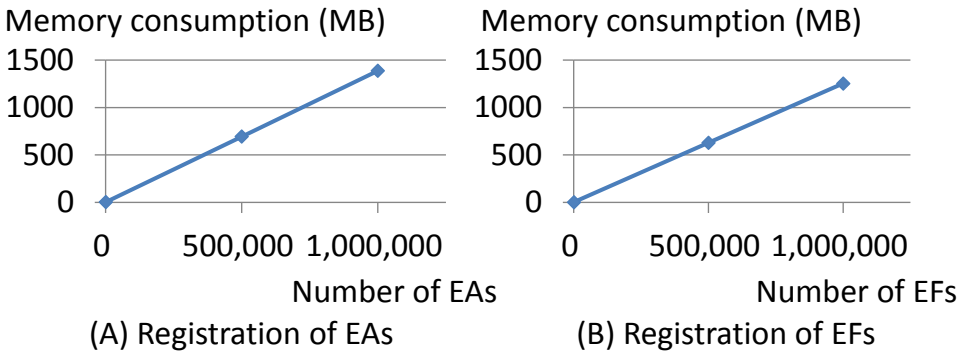


図 52　uTuple の蓄積によるメモリ消費量

4.4.3.1.2 全体性能の見積もり

マイクロベンチマークテストの結果に基づき，2 つ以上の uTuple サーバを含むシステム全体の性能を見積もった．このとき，表 6 に示す 2 種の典型的な M2M アプリケーションの負荷を想定した．アプリケーション A はセンサデータを用い，アプリケーション B は主にアクチュエータを用いる．アプリケーション A はトラックの積み荷の積載量を計測するセンサを用いて，物流の効率を向上させるアプリケーションである．センサデータ蓄積のためのイベント通信負荷は大きいが，コマンド通信負荷は 0 である．センサデバイスを用いてモノ・コトを監視することを主な目的とする M2M アプリケーションの負荷は，このような負荷となるであろう．アプリケーション B は情報機器の遠隔管理のためのアプリケーションである．状態確認のためのイベント通信負荷と制御コマンドを送信するためのコマンド通信負荷がともに大きい．CA は全て sCA である．センサとアクチュエータを用いてモノ・コトを制御することを目的とする M2M アプリケーションの負荷は，このような負荷となるであろう．

表 6　想定アプリケーションの負荷

		Application A	Application B
Event	# of registered EAs	350D	720D
	# of registered EFs	D / 10000	D / 10000
	EA registration freq.	50D (per day) [1]	12D (per day) [1]
	EF registration freq.	0.1 (per day) [1]	0.1 (per day) [1]
Command	# of registered CFs	0	D
	# of registered CAs	0	0
	CF registration freq.	0	1 (per sec) [0]
	CA registration freq.	0	12D (per day) [1]

D: # of devices, [●]: # of matched uTuples in each registration

これら想定負荷に対するシステム全体のスループット（1 秒間に登録可能な uTuple 数）と蓄積可能な uTuple 数を図 53 に示す．図 53 は，最良の状況（Best condition），最悪の状況（Worst condition）それぞれにおいて，従来手法（existing method）と提案手法（proposed method）における性能を見積もったものである．従来手法は，従来の DHT における負荷分散手法であり，同じ分散キーをもつ uTuple 群により生じた負荷を分散させる

ことはできない．uTuple 群の分散キーが一様に分散した状況が最良の状況である．この状況においては，従来手法でも提案手法でも，uTuple 群により生じた負荷は uTuple サーバ群に均等に分散する．一方，uTuple 群の全ての分散キーが同一である状況が最悪の状況である．この状況においては，従来手法では uTuple 群の負荷は 1 つの uTuple サーバに集中してしまう．提案手法では，全ての EF と CA が全 uTuple サーバに複製される．これにより，EF と CA により発生する記憶負荷の増加と，EA と CF を登録する際のマッチングプロセスにおける計算負荷の増加が発生する．これら増加した負荷は全体スループットと蓄積可能 uTuple 数に影響を及ぼす．影響の大きさは図 51 と図 52 から決定することができる．全体スループットと蓄積可能 uTuple 数の見積もりにおいては，図 51 と図 52 に示される結果が線形であり，各種 uTuple は計算リソースおよび記憶リソースをそれぞれ独立に消費し，種類の異なる uTuple が混在した場合にも消費量は各種 uTuple による消費量を足し合わせただけでそれ以上の増加も減少もしないものとした．例えば，EA の登録に 200msec を要し，EF の登録に 100msec を要する場合，EA と EF の登録に要する時間は 300msec であると見積もった．また，各 uTuple サーバは 140GB の記憶容量があると想定し，EF や CA の複製に伴う計算負荷やネットワーク遅延は無視できるものとした．前述したように，分担管理サーバ上で動作する DHT は，各ピアがメモリアクセスのみでクエリ処理を実行できるため高いスループットを示す．例えば，memcached [75]の 1 ノードは秒間約 10 万クエリを処理できると報告されている．オンメモリのピアも同等の性能を達成できるであろう．sCA サーバも，比較的少ない量の情報を保持し，ハッシュテーブル内の検索という単純な処理を実行するため，sCA サーバもやはり同等の性能が期待される．さらに，負荷を DHT により分散させることも可能である．それゆえ，分担管理サーバと sCA サーバは十分に高い性能を示し，uTuple サーバの性能には影響を与えないものとした．また負荷監視サーバからのトラヒックは十分に小さく，uTuple サーバの性能には影響を与えないものとした．

提案手法において，定常状態，つまり動的負荷分散が起きていない状態においては，uTuple サーバ間の通信は発生しない．別の言い方をすれば，uTuple サーバは互いに独立である．uTuple サーバの性能は，他の uTuple サーバの性能や負荷に依存しない．それゆえ，1 つの uTuple サーバ上のマイクロベンチマークの結果から，定常状態における全体スループットと蓄積可能 uTuple 数を見積もることが可能である．

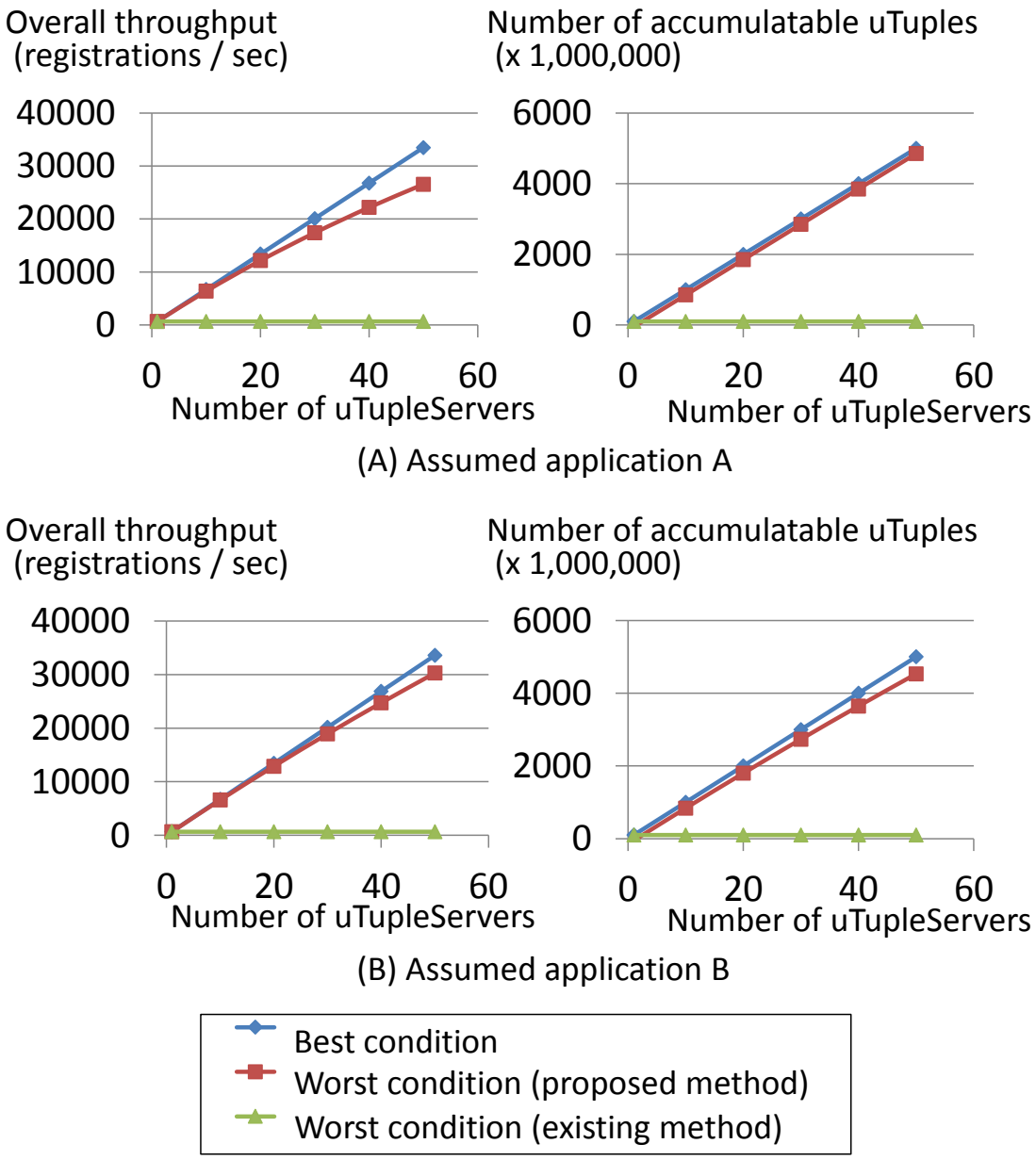


図 53　システム全体の性能

4.4.3.1.3　見積もった性能に関する議論

最良の状況においては，提案手法の動的負荷分散処理は発生せず，従来手法と同じ性能となる．提案手法の効果はより悪い状況，つまり負荷が不均一に分散した場合に現れる．図 53 に見られるように，どちらのアプリケーションの負荷においても，提案手法を用いた場合の性能は最悪の状況において劇的に向上し，最良の状況における性能に近づくことが分かる．

以上より，提案手法は 2 つの典型的な M2M アプリケーションの負荷に対し，たとえ 1 つ

のハッシュ値に負荷が集中した場合にも，小さなオーバーヘッドで負荷を分散させることができると言える．

4.4.3.2 フィージビリティ評価

アプリケーション A の負荷において, uTupleSpace の運用シミュレーションを実施した．性能評価により，提案手法はアプリケーション A，B の負荷を小さなオーバーヘッドで分散できることを確認したが，それだけでは実際のシステム運用に適用することができない．実際のシステム運用においては，スモールスタート，すなわち，数台のサーバを用いて小さな規模でシステムを運用開始し，その後は負荷を監視し，必要に応じてサーバを増設しなければならない．このとき，システムリソースにある程度の余裕を持たせた状態を維持することで，システム管理者が小さな設備投資で安定した運用が可能であることが求められる．そこで，提案手法によりそのような運用ができるかを確認するため，運用シミュレーションを実施した．シミュレーションにおいては，以下の条件を想定した．

- センサデバイスは 1 日につき 1 万台増加し，3 年後には 1000 万台に達する．
- サーバの購入決定から設置までには，購入処理や設定作業などのために 1 ヵ月を要する．

運用ルールは以下のように単純なものとした．

- システムは 10 台の uTuple サーバで運用を開始する．
- システム管理者は，毎日，過去 1 週間の負荷変化から線形予測を行い，60 日後の負荷を予測する．
- 60 日後のリソース使用率が 90%を超えると予測される場合に，システム管理者はリソース使用率を 70%以下にするのに必要な数のサーバの購入を決定する．

シミュレーションの結果を図 54 に示す. グラフは上記運用ルールに従った運用シミュレーションにおける，平均リソース使用率と uTuple サーバの数の変化を示している．図 54 から分かるように，uTuple サーバの数は少しずつ増加し，センサデバイス数が 1000 万台に達する 1000 日目には，uTuple サーバは 50 台に達する．そしてその間，記憶リソース使用率も計算リソース使用率も 50%から 90%の範囲に収まり，平均約 70%を維持している．実際，500 日目から 1000 日目までの記憶リソース使用率と計算リソース使用率の平均はそ

れぞれ 72.8%，66.6%である．

このように，提案手法は 1000 万台のデバイスの負荷に対しても安定性を維持しつつ，低い設備投資に抑える効率的なシステム管理を可能とすると言える．

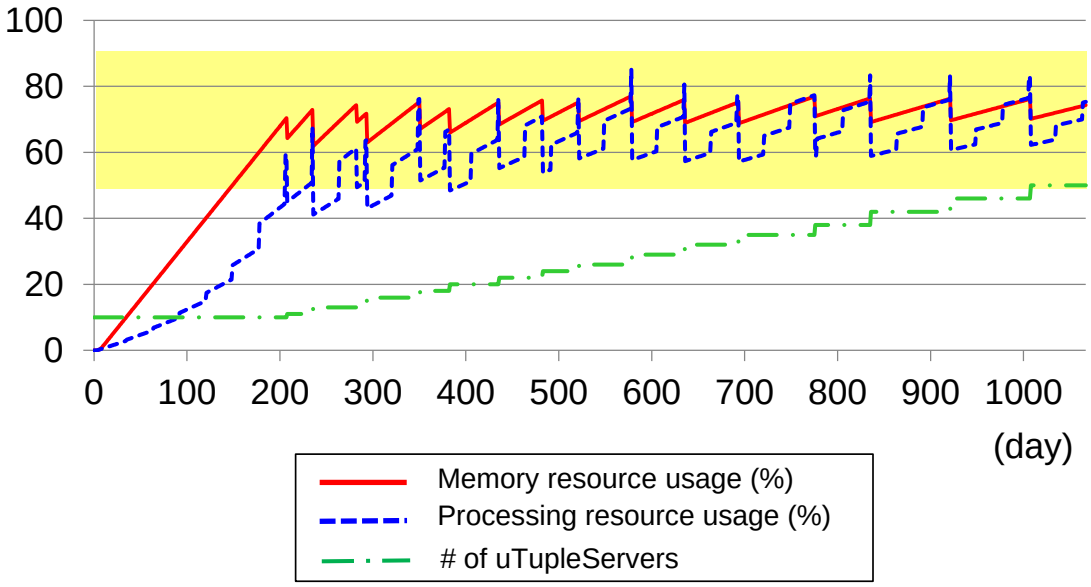


図 54　運用シミュレーションにおける平均リソース使用率と uTuple サーバ数

4.4.4 Dynamic-Help Method まとめ

4.4 では，uTupleSpace の負荷を動的に分散させる，「Dynamic-Help Method」と呼ぶ新しい負荷分散手法を提案した．評価実験により，提案手法はセンサデバイスとアクチュエータデバイスを用いる典型的なアプリケーションの負荷に対して良いスケール性を実現し，また経済的で安定したオペレーションを可能とすることを確認した．

本評価実験においては，uTuple サーバを迅速に実装するため高機能なオープンソースデータベースである PostgreSQL を活用し，マッチングプロセスにおいて必要となる検索の高速化のため PostgreSQL のデフォルトのインデックスである B+-Tree を用いた．UBI-Tree を用いた場合には，前節までで述べたようにさらに高速化が見込まれ，さらに，スキーマレス検索が可能となるため uTuple のデータ形式の自由度を向上させることができ，4.4.2.1 で述べた uTupleSpace モデルよりも柔軟な通信モデルを実現することができると考えられる [76]．また逆に，uTuple サーバにおいて UBI-Tree を用いることで，UBI-Tree は水平統合型プラットフォームのためのスケール性のあるインデキシング技術となると言え

る.

4.5 多次元範囲検索のためのインデキシング技術まとめ

第 4 章では，M2M データの多次元範囲検索のための新しいインデキシング技術「UBI-Tree」について提案し，期待通り提案手法により，スキーマレス検索，すなわち多様な M2M データに対するスキーマ横断的な多次元範囲検索を高速化できることを示した．さらに，負荷変化に動的に適応する追加アルゴリズム（Load-Adaptive UBI-Tree）を提案し，実際に頻繁に用いられるクエリに対する性能が向上し，検索の平均レイテンシやスループットが向上することを示した確認した．また，UBI-Tree を適用する水平分散型プラットフォーム uTupleSpace における新しい動的負荷分散手法，Dynamic-Help Method を提案し，M2M アプリケーションの負荷に対して良いスケール性を実現できることを示した．

提案する UBI-Tree 技術により，高速性とスケール性，負荷追従性をもつ，M2M データに対する多次元範囲検索機能の実現が可能となる．これにより，多くの M2M アプリケーションにより必要とされる，水平統合型プラットフォームに蓄積された多様な M2M データの中から有用なデータを自在に抽出するための，スキーマ横断的な多次元範囲検索が効率的に実現できると言える．

第 5 章

結論

5.1 本論文のまとめ

本論文では，背景として，通信機器の小型化やネットワークインフラストラクチャの大容量化・低料金化などにより M2M アプリケーションが普及しつつあり，更なる普及加速のためには水平統合型プラットフォームの実現が重要であることを述べた．水平統合型プラットフォームは M2M アプリケーション間で共通する機能，あるいはデバイスやデータそのものを共有化し，M2M アプリケーションの開発・運用コスト低減や実装期間短縮を可能とする．

そこで，水平統合型プラットフォームの実現に寄与するため，水平統合型プラットフォームの基本機能の 1 つであるデータ検索機能に好適なインデキシング技術を確立することを本論文の目的とした．当該インデキシング技術には，多様な M2M データとそれに対する M2M アプリケーションの検索クエリに対応可能な，挿入性能，検索性能ともに高速性とスケール性，また負荷追従性をもつ検索機能が求められる．

本論文ではまず，M2M アプリケーションの検索要求について検討・整理した（第 2 章）．例題として市川研究室で検討が進められている蓄電池管理アプリケーションの検索要求について考察し（2.1），これに倣いさらに 14 分野，計 58 種の M2M アプリケーションについて検索要求を検討・整理した．この結果，M2M アプリケーションの検索要求には大きく 5 つの検索パターンが存在することが判明した（2.2）．さらに 5 つの検索パターンの中で，「その他の多次元検索」と分類した検索パターン，すなわち多様な M2M データに対する多次元検索が，これまでにない検索要求であることを述べた（2.3）．「その他の多次元検索」とは，「ID」を検索条件に含む多次元検索や，「位置」と「時間」を検索条件に含む多次元検索以外の多次元検索であり，例えばセンサ値そのもの，あるいはその他属性情報等を検索条件に指定した多次元検索などである．また当該「その他の多次元検索」には多次元完全一致検索と多次元範囲検索が含まれることを述べた．

本論文では次に，多次元完全一致検索の高速性，スケール性，負荷追従性を実現する新しいインデキシング技術として「準候補キーインデキシング」技術を，また多次元範囲検索の高速性，スケール性，負荷追従性を実現する新しいインデキシング技術として「UBI-

Tree」技術を，それぞれ提案し，また評価した．

●準候補キーインデキシング（第 3 章）

第 3 章では，多次元完全一致検索について，高速性，スケール性，負荷追従性を実現する新しいインデキシング技術として，「準候補キーインデキシング」を提案した．スケール性や負荷追従性，また検索条件に自由度のある多次元完全一致検索を可能とすべきという観点から，多次元完全一致検索を行う検索機能は DHT ベースの P2P 検索システムにより実現することが有望である．しかしながら，多次元完全一致検索，すなわち複数の属性情報を指定した検索条件（AND 条件）を用いた検索を行う際，各次元での検索結果（中間解）が膨大となる場合には，中間解の通信オーバーヘッドが大きく，高速性が損なわれてしまうという問題があった．準候補キーインデキシングは，AND 条件を用いた場合にも高速に検索結果を返却できるよう，AND 条件とそれに該当する検索結果も予めインデックスに入れておく手法である．ただし，AND 条件は組合せの数だけ存在し，その数は爆発してしまうため，必要以上に多くの AND 条件をインデキシングしないで済むよう，インデキシングすべき AND 条件の選択手法を考案した点が本技術の重要な部分である．検索結果数が閾値 T 個を越えない AND 条件であり，かつ条件が 1 つでも欠けると検索結果数が T 個以下に絞れない AND 条件を準候補キーと呼び，準候補キーをインデキシングすべき AND 条件とする．準候補キーをインデキシングし，また T 個以下となった検索結果についてはシーケンシャル検索を適用することで，AND で結ばれた条件の数にのみ依存する処理コストで多次元完全一致検索を実現することが可能となる．評価の結果，AND 条件においても M2M データ数に関わらず高速な検索が実現でき，またインデキシングのための通信量や記憶量は M2M データ数に対して log オーダとなり，スケール性があることを確認した．また M2M データの更新に追従して，インデキシングする AND 条件を適応的に変化させていくアルゴリズムを考案し，当該アルゴリズムも許容可能なオーバーヘッドで実現可能であることを確認した．

●UBI-Tree（第 4 章）

第 4 章では，多次元範囲検索について，高速性，スケール性，負荷追従性を実現する新しいインデキシング技術として，「UBI-Tree」を提案した．B-Tree などの 1 次元インデックスを用いる従来の手法では，中間解のマージやフィルタリングに要する処理コストが大きく，高速な多次元範囲検索を行うことができなかった．また R-Tree など従来の多次元インデックスは，同種同数次元のデータや検索条件を前提としており，多種多様な M2M データや検索条件に対して高速な検索処理を行うことができなかった．これに対し UBI-Tree は，異種異数次元を含む M2M データや検索条件を前提とする木構造の多次元インデックスである．UBI-Tree は，各次元が検索条件として用いられる頻度をデータ間の次元

の違いや蓄積データの統計情報から予測し，当該頻度を考慮したインデキシングを行うことで，水平統合型プラットフォームにおいて求められるスキーマ横断的な多次元範囲検索（スキーマレス検索）を高速化する．インデキシング時には，木構造を形成する各ノードの「次元種類数」と「検索時にアクセスされる確率」を考慮し，これらがなるべく小さくなるようにデータを分類する．評価の結果，多次元範囲検索においてもアクセスノード数を抑えることができ，期待通り高速な多次元範囲検索が可能となることを確認した（4.2）．また，UBI-Tree に検索負荷の変化に対する負荷追従性を持たせるため，各次元が検索条件として用いられる頻度の予測にクエリ履歴の統計情報を用いる新しい UBI-Tree，Load-Adaptive UBI-Tree を提案した．評価の結果，実際に頻繁に用いられるクエリ（FQ）に対する性能が向上し，検索機能の平均レイテンシやスループットが向上することを確認した（4.3）．また，UBI-Tree にスケール性を持たせるため，UBI-Tree を適用する水平分散型プラットフォームである uTupleSpace における新しい動的負荷分散手法，Dynamic-Help Method を提案した．uTupleSpace は負荷分散のために DHT ベースの P2P システムを用いているが，従来の DHT は 1 つの分散キーに関係づけられる負荷を複数のピアで分担することができないため，必ずしも十分に負荷を分散することができなかった．Dynamic-Help Method は，「センサデータの数は検索条件数よりも膨大に存在する」，「アクチュエータデバイスは複数回操作される」などの M2M におけるデータや通信の特性を利用することで，分散によるオーバーヘッドを低く抑えるとともに，計算負荷および記憶負荷の特性に応じた負荷分散アルゴリズムをもつ手法である．評価の結果，M2M アプリケーションの負荷に対して良いスケール性を実現できることを確認した（4.4）．

以上より本研究では，水平統合型プラットフォームの検索機能に好適な，高速性，スケール性，負荷追従性を合わせ持つ，多次元検索のための 2 つの新しいインデキシング技術を確立することができたと言える．

5.2 実用化に向けた今後の課題

1.2.1.1 で述べたように，これらのインデキシング技術は，検索要求に応じて使い分ける必要があるが，その使い分け方は必ずしも単純ではない．

例えば準候補キーインデキシング技術は，多次元完全一致検索を効率化し，AND 条件の検索結果が T 個以下でありさえすれば，検索条件の数に依存しない高速な検索を可能とする．しかしながら，「0～10」「10～20」など，値範囲をキーワードとして用いれば，準候補キーインデキシング技術を多次元範囲検索の効率化に用いることもできる．ただし，この際，使用可能な範囲条件は予めキーワードと決めた値範囲に限られるため，範囲条件の自由度

は制限されてしまう．また検索結果が T 個以下となる AND 条件である必要があることも，検索条件に対する自由度の制限となる．T 個以下となる AND 条件をインタラクティブに見つけることが可能なユーザインタフェースがある場合など，適用シーンは限定されると考えられる．

一方，UBI-Tree 技術は多次元範囲検索を効率化するが，始点と終点が同じ値の値範囲を範囲条件とした検索は，多次元完全一致検索であり，多次元完全一致検索の効率化に用いることもできると言える．AND 検索の検索結果が T 個以下でなくとも検索可能であるという利点がある一方，検索結果が膨大となる場合にはそれだけ処理コストは大きくなってしまい，高速性が保証されないという問題がある．

また，2.3 で述べたように，ID など 1 次元だけで十分に検索結果を絞り込める次元を検索条件に含む多次元検索においては，当該 1 次元に対する 1 次元インデックスを用い，中間解をフィルタリングする従来の手法が適している．さらに，多くのインデックスを用いてしまうと，インデックスの構築や維持に要する計算負荷や記憶負荷が大きくなってしまうという問題がある．

今後，本論文で提案したインデキシング技術を活用し，水平統合型プラットフォームを実用化していくためには，他のインデキシング技術や，検索機能以外の機能を含めた水平統合型プラットフォームとしてのトータルな設計，本格実装，実証実験による有効性確認を行っていく必要があると考えられる．

なお，本論文で提案した準候補キーインデキシング技術，UBI-Tree 技術はいずれも，水平統合型プラットフォームにのみ適用可能な技術ではない．それぞれインデキシング技術の１つとして，例えば１つのサーバ上で動作するデータベースにおいても適用可能である．ハッシュテーブル上で AND 検索を高速化したい場合，また XML データベースや，近年注目を集めているスキーマレスデータベースにおける多次元検索を高速化したい場合など，広く適用できる可能性がある．

謝辞

本研究を遂行し学位論文をまとめるにあたり，多くのご支援とご指導を賜りました電気通信大学 大学院情報理工学研究科 教授 市川 晴久 先生に心より感謝申し上げます．ご多忙の中，社会人学生として快く受け入れていただき，また研究の進め方，論文の書き方等大変多くの貴重なご助言を賜りました．

また，学位論文審査において，貴重なご助言を賜りました電気通信大学 大学院情報理工学研究科 教授 太田 和夫 先生，同 吉浦 裕 先生，同 西野 哲朗 先生，同 羽田 陽一 先生に心より感謝申し上げます．

また，入学をお薦めいただき，研究内容へのご助言から履修手続き等のサポートに至るまで，様々な面でお世話になりました電気通信大学 大学院情報理工学研究科 助教 川喜田 佑介 先生に心より感謝申し上げます．

また市川・川喜田研究室の皆様には，ゼミ合宿等で有意義な議論をさせていただくとともに，大変仲良く接していただきました．ありがとうございました．

また，本研究において，各構成技術を作り上げるにあたり，多くのご支援とご指導を賜りました日本電信電話株式会社 未来ねっと研究所の斎藤 洋 様（現在，同社ネットワーク基盤技術研究所），山口 正泰 様（現在，慶應義塾大学 大学院理工学研究科 特任准教授），松村 一 様（現在，NTT アドバンステクノロジ株式会社），松尾 真人 様，中村 元紀 様，南 裕也 様（現在，同社 サービスイノベーション総合研究所），中村 隆幸 様，柏木 啓一郎 様，森 晧平 様，松浦 伸彦 様に心より感謝申し上げます．

また，博士課程に関する活動についてご理解・ご支援いただきました NTT コムウェア株式会社 高橋 英範 様，井藤 雅稔 様，古川 嘉識 様に心より感謝申し上げます．

また，研究を進めるにあたり，ご支援，ご協力を賜りながら，ここにお名前を記すことが出来なかった多くの方々に心より感謝申し上げます．

最後に，日々の生活において，常に心配し労ってくれた父母，そして週末に論文執筆する私のために，共働きにもかかわらず家事を一手に担い，博士課程に関する活動を全面的に応援してくれた妻，祥子に心より感謝します．

引用文献

[1] 野村総合研究所, "2018 年度までの IT 主要市場の規模とトレンドを展望," http://www.nri.com/jp/news/2013/131127.html, accessed January 14, 2014.

[2] Cisco, "Embracing the internet of everything to capture your share of $14.4 trillion," http://www.cisco.com/web/about/ac79/docs/innov/IoE_Economy.pdf, accessed July 13, 2014.

[3] European Commission, " eCall: automated emergency call for road accidents mandatory in cars from 2015," http://ec.europa.eu/digital-agenda/en/news/ecall-automated-emergency-call-road-accidents-mandatory-cars-2015, accessed May 25, 2014.

[4] コマツ建機, "KOMTRAX," http://www.komatsu-kenki.co.jp/service/product/komtrax/, accessed January 14, 2014.

[5] JR 東日本ウォータービジネス, "夢の飲料自販機 エキナカ本格展開へ," http://www.jre-water.com/pdf/100810jisedai-jihanki.pdf, accessed January 14, 2014.

[6] NEC, " CONNEXIVE Products and Services," http://www.nec.com/en/global/solutions/nsp/m2m/prod-sv/index.html, accessed January 14, 2014.

[7] HONDA, "Honda internavi," http://www.honda.co.jp/internavi/, accessed January 14, 2014.

[8] TOYOTA, "G-BOOK," http://g-book.com/pc/default.asp, accessed January 14, 2014.

[9] oneM2M, "oneM2M," http://www.onem2m.org/, accessed January 14, 2014.

[10] 筒井 章博, 後藤 良則, "oneM2M 標準化動向," NTT 技術ジャーナル, Vol. 26, No. 6, pp. 38-41, Jul. 2014.

[11] Ericsson, "Device Connection Platform," http://www.ericsson.com/ourportfolio/products/device-connectionplatform, accessed January 14, 2014.

[12] LogMeIn, Inc., "xively," https://xively.com/, accessed January 14, 2014.

[13] J.A. Burke, D. Estrin, M. Hansen, A. Parker, N. Ramanathan, and S. Reddy, "Participatory sensing," Proc. SenSys'06 Workshop on World Sensor Web, 2006.

[14] N. Kodama and Y. Hada, "Road information gathering and sharing during disasters using probe vehicles," International Journal of ITS Research, vol.6, no.2, pp.97-104, December 2008.

[15] Weathernews Inc., "ウェザーニュース：ウェザーリポート Ch.," http://weathernews.jp/, accessed May 25, 2014.

[16] 鈴木 幸市, 藤塚 勤也, "RDBMS 解剖学 よくわかるリレーショナルデータベースの仕組み," pp. 4-8, 翔泳社, 2005.

[17] 小島 功士, 竹本 大輔, 田頭 茂明, 藤田 聡, "P2P システム上でのリザルトキャッシングを用いた条件付検索手法," 信学技法, IN2004-124, pp.7-12, Dec. 2004.

[18] M.K. Aguilera, W. Golab, and M.A. Shah, "A practical scalable distributed B-tree," Proc. of VLDB Endowment, Vol. 1, No. 1, pp. 598-609, Aug. 2008.

[19] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels, "Dynamo: Amazon's highly available key-value store," Proc. of ACM SIGOPS, Vol. 41, No. 6, pp. 205–220, Dec. 2007.

[20] T. Abraham and J.F. Roddick, "Survey of Spatio-Temporal Databases," Journal Geoinformatica, Vol. 3, No. 1, pp. 61-99, Mar. 1999.

[21] N. Pelekis, B. Theodoulidis, I. Kopanakis, and Y. Theodoridis, "Literature review of spatio-temporal database models," The Knowledge Engineering Review, Cambridge University Press, Vol. 19, No. 3, pp 235-274, Sep. 2004.

[22] T.B. Mishra and S. Sahni, "PUBSUB: An efficient publish/subscribe system," IEEE Trans. on Computers, Apr. 2014.

[23] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: semantic foundations and query execution," The VLDB Journal, Vol. 15, No. 2, pp. 121-142, Jun. 2006.

[24] A. Bainbridge, E. Bouillet, Z. Nabi, and C. Thomas, "Of Streams and Storms: a direct comparison of IBM InfoSphere Streams and Apache Storm in a Real-World Use Case - Email Processing," IBM Research Dublin and IBM Software Group

Europe white paper, Apr. 2014.

[25] 中村 隆幸, 柏木 啓一郎, 荒川 豊, 森 皓平, 松浦 伸彦, 中村 元紀, "多次元検索木 UBI-Tree を用いた uTupleSpace のアーキテクチャ," 2013 信学総大, B-19-28, Mar. 2013.

[26] The Distribution Systems Research Institute, "EPCglobal Japan について," http://www.dsri.jp/epcgl/epc/about_epcglj.htm, accessed May 25, 2014.

[27] S. Shi, G. Yang, D. Wang, J. Yu, S. Qu, M. Chen, "Making peer-to-peer keyword searching feasible using multi-level partitioning," Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), San Diego, USA, Feb. 2004.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp and S. Shenker, "A scalable content-addressable network," Proc. ACM SIGCOMM 2001, pp.161-172, San Diego, USA, Aug. 2001.

[29] I. Stoica, R. Morris, D. Karger, M. Frans Kaashoek and H. Balakrishnan, "Chord: a scalable peer-to-peer lookup service for internet applications," Proc. ACM SIGCOMM 2001, pp.149-160, San Diego, USA, Aug. 2001.

[30] A. Rowstron and P. Druschel, "Pastry: scalable, distributed object location and routing for largescale peer-to-peer systems," Proc. 18th IFIP/ACM International Conf. on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany, Nov. 2001.

[31] B.Y. Zhao, J. Kubiatowicz, and A.D. Joseph, "Tapestry: an infrastructure for fault-tolerant wide-area location and routing," Tech. Rep. UCB/CSD-01-1141, UC Berkeley, Apr. 2000.

[32] 速水 賢史, 竹野 浩, 永瀬 智哉, 藤本 典幸, 萩原 兼一, "スケーラビリティのある WWW 並列全文検索システム構築法の提案と評価," 情処学データベース研報, no.123, pp.45-52, 2000.

[33] B.T. Loo, R. Huebsch, I. Stoica and J.M. Hellerstein, "The case for a hybrid P2P search infrastructure," Proc. 3rd International Workshop on Peer-to-Peer Systems (IPTPS'04), San Diego, USA, Feb. 2004.

[34] http://dblp.uni-trier.de/

[35] P. Reynolds and A. Vahdat, "Efficient peer-to-peer keyword searching," Proc. Middleware 2003, pp.21-40, Jun. 2003.

[36] I.H. Witten, A. Moffat and T.C. Bell, "Managing Gigabytes: Compressing and Indexing Documents and Images," Van Nostrand Reinhold, New York, 1999.

[37] E.D. Demaine, A. Lopez-Ortiz and J.I. Munro, "Adaptive set intersections, unions, and differences," Proc. 11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2000), San Francisco, USA, Jan. 2000.

[38] J. Li, B.T. Loo, J.M. Hellerstein, M.F. Kaashoek, D.R. Karger and R. Morris, "On the feasibility of peer-to-peer web indexing and search," Proc. 2nd International Workshop on Peer-to-Peer Systems (IPTPS'03), Berkeley, USA, Feb. 2003.

[39] O.D. Gnawali, "A keyword-set search system for peer-to-peer networks," Master's thesis, Massachusetts Institute of Technology, Jun. 2002.

[40] B.T. Loo, J.M. Hellerstein, R. Huebsch, S. Shenker and I. Stoica, "Enhancing P2P file-sharing with an internet-scale query processor," Proc. 30th International Conf. on Very Large Data Bases (VLDB 2004), Toronto, Canada, Sep. 2004.

[41] The Apache Software Foundation, "Apache Couch-DB: The Apache CouchDB Project," http://couchdb.apache.org/, 参照 June 2013.

[42] 10gen, Inc., "MongoDB," http://www.mongodb.org/, 参照 June 2013.

[43] A. Guttman, "R-Trees: A dynamic index structure for spatial searching," Proc. ACM SIGMOD, pp.47-57, 1984.

[44] J. McHugh and J. Widom, "Query Optimization for XML," Proc. 25th VLDB, pp.315-326, 1999.

[45] X. Dong and A. Halevy, "Indexing Dataspaces," Proc. ACM SIGMOD, pp.43-54, 2007.

[46] J.L. Bentley, "K-d trees for semidynamic point sets," Proc. 6th Annual Symposium on Computational Geometry, pp.187-197, 1990.

[47] T. Sellis, N. Roussopoulos, and C. Faloutsos, "The R+-tree: a dynamic index for multi-dimensional objects," Proc. 13th VLDB, pp.507-518, 1987.

[48] N. Beckmann and H.P. Kriegel. "The R*-tree: an efficient and robust access method for points and rectangles," Proc. ACM SIGMOD, pp.322-331, 1990.

[49] N. Katayama and S. Satoh, "The SR-tree: an index structure for high-dimensional nearest neighbor queries," Proc. ACM SIGMOD Int. Conf. on Management of Data, pp.369-380, 1997.

[50] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," Communications of the ACM, vol.51, no.1, pp.117-122, 2008.

[51] H.V. Jagadish, B.C. Ooi, K.L. Tan, C. Yu, and R. Zhang, "iDistance: an adaptive B+-tree based indexing method for nearest neighbor search," Proc. ACM TODS, vol.30, no.2, pp.364-397, June 2005.

[52] P.A. Burrough and R.A. McDonnell, "Principles of geographical information systems," Oxford, 1998.

[53] R. Wever, H.J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," Proc. 24th VLDB, pp.194-205, 1998.

[54] K. Chakrabarti and S. Mehrotra, "Local dimensionality reduction: a new approach to indexing high dimensional spaces," Proc. 26th VLDB, pp.89-100, 2000.

[55] J.M. Hellerstein, J.F. Naughton, and A. Pfeffer, "Generalized Search Trees for Database Systems," Proc. 21st Int. Conf. on Very Large Data Bases, pp.562-573, September 1995.

[56] S. Fushimi, M. Kitsuregawa, M. Nakayama, H. Tanaka, and T. Motooka, "Algorithm and Performance Evaluation of Adaptive Multidimensional Clustering Technique," Proc. of ACM SIGMOD, pp.308-318, 1985.

[57] 大森匡, 佐藤龍生, 星守, "問合せ分布を考慮した R 木における領域分割方式," 電子情報通信学会論文誌, vol. J86-D-I, no. 10, pp. 746-761, Oct. 2003.

[58] Live E!, "Live E! Environmental information for a living earth," http://www.live-e.org/en/index.html, accessed January 8, 2014.

[59] T. Nakamura, M. Nakamura, A. Yamamoto, K. Kashiwagi, Y. Arakawa, M. Matsuo, and H. Minami, "uTupleSpace: A bi-directional shared data space for wide-area sensor network," Proc. of 2nd Intl. Workshop on Sensor Networks and Ambient Intelligence (SeNAmI 2009), pp.396-401, December 2009.

[60] D. Gelernter, "Generative Communication in Linda," ACM Transactions on Programming Language and Systems, vol. 7, no. 1, pp. 80-112, Jan. 1985.

[61] C. Curino, M. Giani, M. Giorgetta, A. Giusti, A.L. Murphy, and G.P. Picco, "TinyLIME: Bridging Mobile and Sensor Networks through Middleware," Proc. International Conference on Pervasive Computing and Communications (PerCom

'05), pp. 61-72, 2005.

[62] P. Costa, L. Mottola, A.L. Murphy, and G.P. Picco, "TeenyLIME: Transiently Shared Tuple Space Middleware for Wireless Sensor Networks," Proc. International Workshop on Middleware for Sensor Networks (MidSens '06), pp. 43-48, 2006.

[63] G. Castelli, A. Rosi, M. Mamei, and F. Zambonelli, "A Simple Model and Infrastructure for Context-aware Browsing of theWorld," Proc. International Conference on Pervasive Computing and Communications (PerCom'07), pp. 229-238, 2007.

[64] G. Hackmann, C.L. Fok, G.C. Roman, and C. Lu, "Agimone: Middleware Support for Seamless Integration of Sensor and IP Networks," Proc. International Conference on Distributed Computing in Sensor Systems (DCOSS '06), pp. 101-118, 2006.

[65] C.L. Fok, G.C. Roman, and C. Lu, "Rapid Development and Flexible Deployment of Adaptive Wireless Sensor Network Applications," Proc. International Conference on Distributed Computing Systems (ICDCS '05), pp. 653-662, 2005.

[66] C.L. Fok, G.C. Roman, and G. Hackmann, "A Lightweight Coordination Middleware for Mobile Computing," LNCS (COORDINATION 2004), vol. 2949, pp. 135-151, 2004.

[67] S. De, S. Nandi, and D. Goswami, "On Performance Improvement Issues in Unordered Tuple Space based Mobile Middleware," Proc. 2010 Annual IEEE India Conference (INDICON 2010), Dec. 2010.

[68] Y. Jiang, G. Xue, Z. Jia, and J. You, "DTuples: A Distributed Hash Table based Tuple Space Service for Distributed Coordination," Grid and Cooperative Computing, Fifth International Conference, pp. 101-106, Oct. 2006.

[69] U. Wickramasinghe, C. Wickramarachchi, P. Fernando, D. Sumanasena, S. Perera, and S. Weerawarana, "BISSA: Empowering Web gadget Communication with Tuple Spaces," Proc. Gateway Computing Environments Workshop (GCE), Nov. 2010.

[70] A. Rao, K. Lakshminarayanan, S. Surana, R. Karp, and I. Stoica, "Load Balancing in Structured P2P Systems," in Proc. 2nd Int'l Workshop Peer-to-Peer Systems (IPTPS'02), pp. 68-79, Feb. 2003.

[71] S. Surana, B. Godfrey, K. Lakshminarayansan, R. Karp, and I. Stoica, "Load Balancing in Dynamic Structured P2P Systems," Performance Evaluation, vol. 63,

no. 6, pp. 217-240, Mar. 2006.

[72] C. Chen and K.C. Tsai, "The Server Reassignment Problem for Load Balancing in Structured P2P Systems," IEEE Trans. Parallel Distrib. Syst., vol. 12, no. 2, pp. 234-246, Feb. 2008.

[73] L. Yang and Z. Chen, "A VS-split Load Balancing Algorithm in DHT-Based P2P Systems," Proc. on 2012 International Conference on Systems and Informatics (ICSAI 2012), pp. 1581-1585, 2012.

[74] H.C. Hsiao, "Load Balance with Imperfect Information in Structured Peer-to-Peer Systems," IEEE Trans. on Parallel and Distributed Systems, vol. 22, no. 4, pp. 634-649, Apr. 2011.

[75] Dormando, "memcached," http://memcached.org/, accessed Oct. 25, 2013.

[76] Takayuki Nakamura, Keiichiro Kashiwagi, Yutaka Arakawa, and Motonori Nakamura, "Design and implementation of new uTupleSpace enabling storage and retrieval of large amount of schema-less sensor data," Proc. of the IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT 2011), pp. 414-420, Jul. 2011.

関連発表

論文

荒川 豊, 南 裕也, 松尾 真人, 山口 正泰, 斎藤 洋, "準候補キーインデキシングを用いた DHT ベース P2P 検索システム," 電子情報通信学会論文誌, Vol.J88-B, No.11, pp.2158-2170, Nov. 2005.

中村 隆幸, 荒川 豊, 山本 淳, 柏木 啓一郎, 東島 由佳, 南 裕也, 中村 元紀, 松尾 真人, "ユビキタスデータ共有機構 uTupleSpace の提案とフィールド実験への適用評価," 電子情報通信学会論文誌, Vol.J95-B, No.11, pp.1414-1426, Nov. 2012.

荒川 豊, 中村 隆幸, 森 皓平, 中村 元紀, "UBI-Tree: スキーマレス検索のためのインデキシング手法," 電子情報通信学会論文誌, Vol. J97-D, No. 4, pp. 807-821, Apr. 2014.

Yutaka Arakawa, Keiichiro Kashiwagi, Takayuki Nakamura, and Motonori Nakamura, "Dynamic Load-Distribution Method of uTupleSpace Data-Sharing Mechanism for Ubiquitous Data," IEICE Transactions, Information & Systems, Vol. E97-D, No. 4, pp. 644-653, Apr. 2014.

国際学会（査読付き国際会議）

Takayuki Nakamura, Motonori Nakamura, Atsushi Yamamoto, Keiichiro Kashiwagi, Yutaka Arakawa, Masato Matsuo, and Hiroya Minami, "uTupleSpace: A bi-directional shared data space for wide-area sensor network," Proc. of 2nd International Workshop on Sensor Networks and Ambient Intelligence (SeNAmI 2009), pp. 396-401, Dec. 2009.

Yutaka Arakawa, Keiichiro Kashiwagi, Takayuki Nakamura, Motonori Nakamura, and Masato Matsuo, "Dynamic scaling method of uTupleSpace data-

sharing mechanism for wide area ubiquitous network," Proc. of Asia-Pacific Symposium on Information and Telecommunication Technologies (APSITT), pp. 1-6, Jun. 2010.

Takayuki Nakamura, Keiichiro Kashiwagi, Yutaka Arakawa, and Motonori Nakamura, "Design and implementation of new uTupleSpace enabling storage and retrieval of large amount of schema-less sensor data," Proc. of the IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT 2011), pp. 414-420, Jul. 2011.

Takayuki Nakamura, Yukio Kikuya, Yutaka Arakawa, Motonori Nakamura, Yuka Higashijima, Yasuko Yamada Maruo, and Masayuki Nakamura, "Proposal of web framework for ubiquitous sensor network and its trial application using NO2 sensor mounted on bicycle," Proc. of the IEEE/IPSJ 12th International Symposium on Applications and the Internet (SAINT 2012), pp. 83-90, Jul. 2012.

Yutaka Arakawa, Takayuki Nakamura, Motonori Nakamura, and Hajime Matsumura, "UBI-Tree: Indexing method for schema-less search," Proc. of the IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW 2013), pp. 283-288, Jul. 2013.

Yutaka Arakawa, Takayuki Nakamura, Motonori Nakamura, Nobuhiko Matsuura, Yuusuke Kawakita, and Haruhisa Ichikawa, "Load-adaptive indexing method for schema-less searches," Proc. of the IEEE 38th Annual Computer Software and Applications Conference Workshops (COMPSACW 2014), pp. 318-324, Jul. 2014.

国内学会

南 裕也, 荒川 豊, 井出 一郎, 斎藤 洋, "準候補キーを用いた物品情報検索システムの提案," 2004 信学ソ大, B-7-62, Sep. 2004.

荒川 豊, 南 裕也, 井出 一郎, 斎藤 洋, "準候補キー方式インデキシングアルゴリ

ズムの提案," 2004 信学ソ大, B-7-63, Sep. 2004.

荒川 豊, 南 裕也, 松尾 真人, 山口 正泰, "AND 検索のための高効率インデックスの提案," 信学技報, vol. 104, no. 516, NS2004-179, pp. 35-38, Dec. 2004.

東條 弘, 高杉 耕一, 神谷 弘樹, 柴田 弘, 小田部 悟士, 石塚 美加, 俊長 秀紀, 南 裕也, 山本 淳, 荒川 豊, 松尾 真人, 石原 晋也, 斎藤 洋, "広域ユビキタスネットワーク構成技術とプラットフォーム技術の提案," 2008 信学総大, B-7-141, Mar. 2008.

山本 淳, 荒川 豊, 南 裕也, 松尾 真人, 東條 弘, 斎藤 洋, "広域ユビキタスプラットフォームの提案," 2008 信学総大, B-7-147, Mar. 2008.

南 裕也, 荒川 豊, 山本 淳, 松尾 真人, 東條 弘, 斎藤 洋, "広域ユビキタスネットワークの特徴を考慮したプラットフォーム構成," 2008 信学総大, B-7-148, Mar. 2008.

荒川 豊, 山本 淳, 南 裕也, 松尾 真人, 東條 弘, 斎藤 洋, "スケーラビリティを考慮した広域ユビキタスプラットフォーム実装方式," 2008 信学総大, B-7-149, Mar. 2008.

荒川 豊, 柏木 啓一郎, 中村 隆幸, 中村 元紀, 松尾 真人, "広域ユビキタスプラットフォームの動的スケール化," 2009 信学総大, B-7-7, Mar. 2009.

荒川 豊, 柏木 啓一郎, 中村 隆幸, 中村 元紀, 松尾 真人, "実世界データ共有機構 uTupleSpace における動的スケール化方式の評価," 信学技報, Vol. 109, no. 79, IN2009-21, pp. 49-54, Jun. 2009.

荒川 豊, 中村 隆幸, 中村 元紀, 松尾 真人, "UBI-tree: ユビキタスデータのためのインデキシング技術," 2010 情処全大, 5C-1, Mar. 2010.

中村 隆幸, 柏木 啓一郎, 荒川 豊, 中村 元紀, "かたまり生成処理により効率化し

たセンサ情報蓄積システムの提案," 2010 信学総大, B-20-3, Mar. 2010.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, 松尾 真人, "大量スキーマレスデータの蓄積・検索を実現する新しい uTupleSpace の設計と実装," DICOMO2010 シンポジウム, pp. 76-82, Jul. 2010.

荒川 豊, 中村 隆幸, 中村 元紀, 松尾 真人, "ユビキタスデータのためのインデキシング技術 UBI-tree の改良," 信学技報, vol. 110, no. 162, DE2010-22, pp. 47-52, Aug. 2010.

中村 隆幸, 東島 由佳, 荒川 豊, 中村 元紀, "ユビキタス向け Web フレームワークの設計と実装," 信学技報, vol. 110, no. 378, USN2010-41, pp.19-24, Jan. 2011.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "ユビキタスデータ共有機構 uTupleSpace における効率的なアクセス制御方式の提案," 信学技報, vol. 110, no. 449, IN2010-184, pp.241-245, Mar. 2011.

森 皓平, 柏木 啓一郎, 中村 隆幸, 荒川 豊, 東島 由佳, 中村 元紀, "uTupleSpace を用いたセンサ利用サービスの開発を支援する「実世界開発スタジオ」の提案," 2011 信学総大, B-20-51, Mar. 2011.

中村 元紀, 中村 隆幸, 荒川 豊, 東島 由佳, 柏木 啓一郎, 森 皓平, 松村 一, 石田 繁巳, 猿渡 俊介, 翁長 久, 森川 博之, "uTupleSpace を利用した CO2 排出量可視化の実証実験," 2011 信学ソ大, B-19-21, Sep. 2011.

中村 隆幸, 荒川 豊, 柏木 啓一郎, 森 皓平, 中村 元紀, "ユビキタスデータ共有機構 uTupleSpace における新しいチャンク形式と高速探索," 2011 信学ソ大, B-19-23, Sep. 2011.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "ユビキタスデータ共有機構 uTupleSpace における効率的なアクセス制御方式の実装と評価," 2011 信学ソ大, B-19-24, Sep. 2011.

森 皓平, 中村 隆幸, 柏木 啓一郎, 荒川 豊, 中村 元紀, 松村 一, 東島 由佳, 石田 繁巳, 猿渡 俊介, 翁長 久, 森川 博之, "uTupleSpace を用いたセンサ利用サービスの開発を支援する「実世界開発スタジオ」の提案と評価," 信学技報, vol. 111, no. 386, USN2011-72, pp.89-94, Jan. 2012.

荒川 豊, 中村 隆幸, 中村 元紀, "ユビキタスデータのためのインデキシング技術 UBI-Tree における検索コスト見積もり方式の改良," 第 4 回データ工学と情報マネジメントに関するフォーラム(DEIM 2012), F11-1, Mar. 2012.

荒川 豊, 中村 隆幸, 森 皓平, 中村 元紀, "多次元検索木 UBI-Tree を用いたスキーマレスなセンサデータの高品質なチャンク生成方式," 2012 信学総大, B-19-16, Mar. 2012.

中村 隆幸, 荒川 豊, 森 皓平, 中村 元紀, "2 つの UBI-Tree による uTupleSpace の省メモリなデータ保管方式," 2012 信学総大, B-19-17, Mar. 2012.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "ユビキタスデータ共有機構 uTupleSpace における改良アクセス制御方式の実装と評価," 信学技報, vol. 112, no. 31, USN2012-13, pp.99-104, May, 2012.

荒川 豊, 中村 隆幸, 中村 元紀, 松村 一, "多次元検索木 UBI-Tree における簡約化データ構造の提案," 2012 信学ソ大, B-19-1, Sep. 2012.

森 皓平, 荒川 豊, 中村 隆幸, 中村 元紀, 松村 一, Ulrich Meis, 計 宇生, "ユビキタスデータ向けインデキシング技術 UBI-Tree の可視化," 2012 信学ソ大, B-19-2, Sep. 2012.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "uTupleSpace 問い合わせライブラリの設計と実装," 2012 信学ソ大, B-19-7, Sep. 2012.

森 皓平, 荒川 豊, 中村 隆幸, 中村 元紀, 松村 一, "UBI-Tree を用いたスキーマ

レスデータの高品質なチャンク生成方式の提案と評価," 信学技報, vol. 112, no. 406, USN2012-62, pp.33-38, Jan. 2013.

森 皓平, 荒川 豊, 中村 隆幸, 中村 元紀, 松村 一, "パターン木による uTupleSpace のスキーマレスデータ可視化手法," 2013 信学総大, B-19-27, Mar. 2013.

中村 隆幸, 柏木 啓一郎, 荒川 豊, 森 皓平, 松浦 伸彦, 中村 元紀, "多次元検索木 UBI-Tree を用いた uTupleSpace のアーキテクチャ," 2013 信学総大, B-19-28, Mar. 2013.

松浦 伸彦, 荒川 豊, 中村 隆幸, 中村 元紀, 松村 一, "UBI-Tree を用いたチャンクデータ生成方式による検索性能の評価," 2013 信学総大, B-19-29, Mar. 2013.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "軽量化 uTupleSpace の設計と実装," 2013 信学総大, B-19-30, Mar. 2013.

松浦 伸彦, 荒川 豊, 中村 隆幸, 中村 元紀, "uTupleSpace における効率的なデータ並行処理手法," 2013 信学ソ大, B-18-4, Sep. 2013.

柏木 啓一郎, 荒川 豊, 中村 隆幸, 中村 元紀, "uTupleSpace におけるデータ公開条件を用いた効率的なアクセス制御の実現," 2013 信学ソ大, B-18-5, Sep. 2013.

特許

荒川 豊, 南 裕也, 特許 4444807, "リソース検索装置，リソース検索方法およびリソース検索プログラム"

松尾 真人, 山本 淳, 南 裕也, 荒川 豊, 特許 4890412, "ネットワーク分散共有システム，ネットワーク分散共有方法，およびネットワーク分散共有プログラム"

松尾 真人, 山本 淳, 南 裕也, 荒川 豊, 特許 4902559, "ネットワーク分散共有シ

ステム，ネットワーク分散共有方法，およびネットワーク分散共有プログラム"

荒川 豊, 南 裕也, 山本 淳, 松尾 真人, 特許 5132359, "データ分散処理システム及び方法"

山本 淳, 荒川 豊, 南 裕也, 松尾 真人, 特許 4820833, "間接通信システム，通信方法および通信プログラム"

荒川 豊, 柏木 啓一郎, 中村 隆幸, 特許 5203253, "タプル蓄積・検索システム，タプル蓄積・検索方法，タプル装置及びタプル分配装置"

荒川 豊, 中村 隆幸, 中村 元紀, 特許 5430436, "情報蓄積検索方法及び情報蓄積検索プログラム"

荒川 豊, 中村 隆幸, 中村 元紀, 特開 2011-170461, "情報蓄積検索方法及び情報蓄積検索プログラム"

中村 隆幸, 荒川 豊, 特開 2011-170791, "情報記録装置，情報記録方法およびプログラム"

荒川 豊, 中村 隆幸, 柏木 啓一郎, 中村 元紀, 特許 5467002, "間接通信装置，通信システム，通信方法および通信プログラム"

荒川 豊, 中村 隆幸, 特開 2013-8295, "情報記録装置，情報記録方法およびプログラム

中村 隆幸, 荒川 豊, 特開 2013-016112, "チャンク生成装置，チャンク読み取り装置，チャンク生成方法及びプログラム"

中村 隆幸, 荒川 豊, 特開 2013-218490, "情報蓄積検索装置"

荒川 豊, 特開 2014-041438, "データ索引装置，データ索引方法及びデータ索引プログラム"

森 皓平, 中村 隆幸, 荒川 豊, 計 宇生, ウルリッヒ アルビノ メイス, 特開 2014-041504, "検索木描画装置，検索木描画方法およびプログラム"

森 皓平, 中村 隆幸, 荒川 豊, 特願 2013-042202, "情報可視化装置, 方法及びプログラム"

荒川 豊, 中村 隆幸, 松浦 伸彦, 特願 2013-099231, "木構造索引を用いたデータ索引装置，データ索引方法およびプログラム"

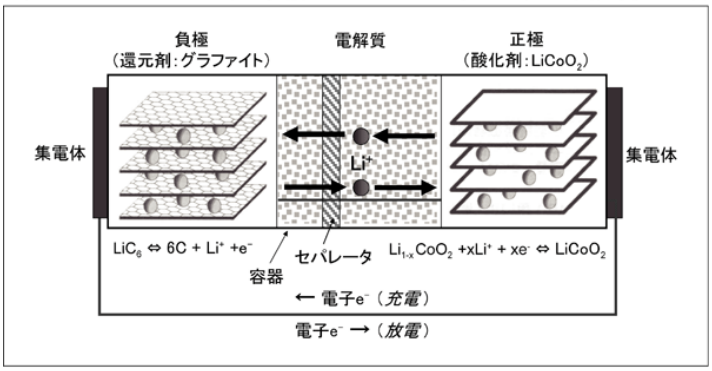
荒川 豊, 中村 隆幸, 松浦 伸彦, 出願手続き中, "木構造索引を用いたデータ索引装置，データ索引方法およびプログラム"

付録A Li イオン電池管理に関する資料

Liイオン電池管理

1

図表２　リチウムイオン電池の充放電メカニズムと電極における電気化学反応の例



科学技術動向研究センターにて作成

現在実用化されているリチウム二次電池の電池は、正極活物質にはLiCoO2などの遷移金属酸化物が、負極活物質には黒鉛などの炭素材料が、電解質は有機溶媒にLi塩を溶かした非水溶液が使用されています。充電時には正極材の結晶構造内にあるリチウムイオンが引き抜かれ、電解質を経由して負極材のカーボンのグラファイト層間へ挿入されます。一方、電気的中性を維持するため外部回路を経由して電子が正極から負極へと移動します。放電時にはこの逆反応が進行し、負極材から正極材へとリチウムイオンと電子の移動が起きます。

履歴DBを用いた電池の管理

- 利用
  - SOC推定
  - 充放電制御：長寿命化、安全確保
- 再利用
  - 電池の評価
- 安全確保
  - 事故防止
  - 事故原因分析

5

電池寿命を決めるパラメータ

- 電池の寿命（＝容量低下）を決める要因
  - 充放電劣化：充放電時の正負極材料の膨張収縮による劣化
  - 稼働時間劣化：電池内部の化学反応による劣化。充放電しなくても進行する
  - 温度：充放電劣化、稼働時間劣化はともに温度が高くなると加速する

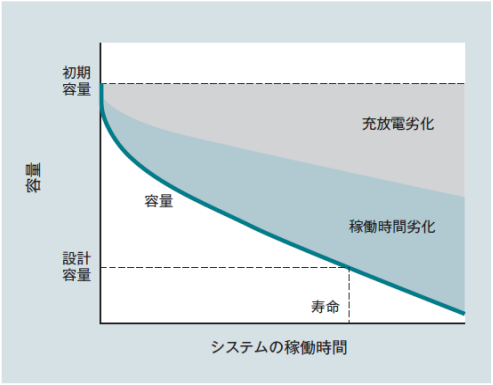


図1｜電池寿命の定義と容量低下
電池は充放電を繰り返すと容量が低下する。電池の容量低下は，充放電劣化と稼働時間劣化がおのおの独立して進行すると考えた。

7

Liイオン電池の安全性確保

9

Liイオン電池事故原因の多くは過充電

- 電池自身の欠陥による事故発生率は極めて低い
  - 2009年3月に米国フロリダナト州で開催された電池の国際学会“26th International Battery Seminar &Exhibit”において、米SandiaNational Laboratory (SNL)は、電池自身が原因となる事故の確率はIppm以下と報告
  - 実際のLiイオン電池事故はppmオーダよりもはるかに高い確率で発生しており、SNLは、事故原因のほとんどが圧壊や加熱、過充電などの外的要因であると報告
- 電池の通常レベルの充電中に、電極にデンドライト状の金属が生成されて内部ショートが発生する要因
  - 電極中に紛れ込んだ金属片が電解液中に溶出して対極に析出
  - 数100ppmの多量の水分が電池内部に含まれた場合、支持電解質と反応してフッ化水素酸(HF)を生成。HFは強酸なので正極活物質や集電体を溶出させ、充放電の際にこれらを対極に金属としてデンドライト状に析出させることがある。電池製造工程では徹底的な水分除去対策を実施のが通例。

次世代電池2011-2012, 日経BP

11

2次電池に過充電は大敵

- 圧縮、くぎ刺ししした電池を充電する試験（圧壊試験、くぎ刺し試験）
  - 4.25V充電では問題なし
  - 4.30V充電では発煙・発火に至る

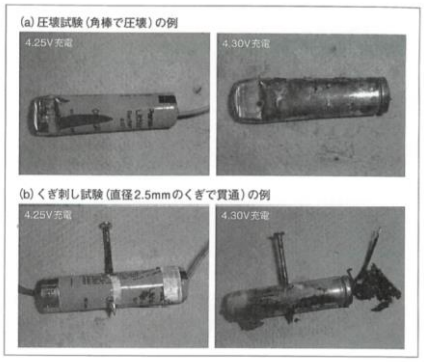


図1 電池が燃える大きな原因の一つが過充電
充電電圧がわずか50mV程度違うだけで、電池は燃えることがある（a,b）。（写真：富士通テクノリサーチ）

次世代電池2011-2012, 日経BP

13

過充電が危険な2つの理由

- 負極のLi収容能力（理論容量は黒鉛で372mAh/g）を超えたLiが金属Liとなり、デンドライト状に析出する
- 正極から反応性が非常に高い酸素が発生する
  - 正極の代表例であるコバルト酸リチウム（$LiCoO_2$）は、CoO2の層の間にLiが挿入された構造。充電が進むとLiが正極から抜けてLiの場所にCoが入り込み、CoO_2の形でCoに結び付いていた酸素（"発生期の酸素"）は、相手がいなくなり正極から脱離する。
  - この酸素が酸化されやすい電解液やLiとCの化合物と反応して発熱、発火することがある。
  - 発生期の酸素は充電が進むほど低温で発生し、量も多くなる

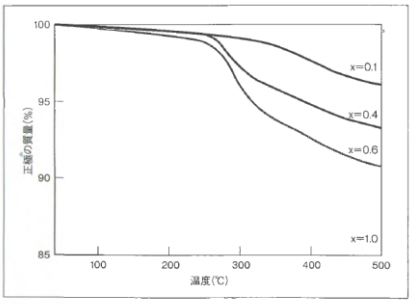


図2 充電が進むほど正極質量は低下
$Li_{1-x}CoO_2$のxの違いによる正極の質量変化を示した。xが大きくなるほど、充電が進んでいることを表す。酸素の脱離によって、正極の質量は小さくなる。

次世代電池2011-2012, 日経BP

15

過充電になる原因

- 急速充電
  - Liイオン電池の推奨充電法である定電流/定電圧充電では、一定の電流で充電し、規定電圧に達すると一定電圧で充電する
  - 定電流充電で充電電圧が規定電圧になっても端子電圧は規定電圧になっていない。低電圧充電では電流が絞られて満充電までに時間がかかる
  - 定電流充電で高い充電率を達成しようとすると規定電圧を超える電圧で充電する(過充電になる)ことになる
- 組電池における電池セルの不均一充電
  - 2セル直列組電池による説明
    - 2セルのうち一方の容量が大きく劣化しているとする
    - 充電すると、容量劣化したセルの方がもう一方のセルよりも早く満充電状態になる
    - 例えば、一方の電圧は4.2V、もう一方は3.8V。充電電圧は4.2Vの2倍の8.4Vをかけるので一方が4.4Vで他方が4.0Vで充電を終了する
    - 4.4Vのセルは過充電状態
  - 対策
    - 保護回路をつけて、規定電圧に達したセルの充電を停止する
    - 過充電を防げる材料を開発する：正極で発生した酸素を負極で消費するような材料。このような反応をレドックス(Redox)という。還元(Reduction)と酸化(Oxidation)の合成語。
    - 燃えない電解液の開発

次世代電池2011-2012, 日経BP

17

Liイオン電池の残量推定

19

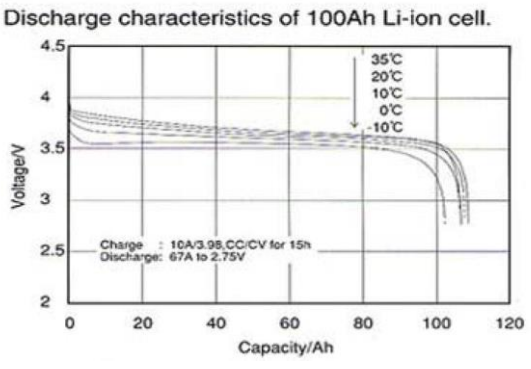
電池残量を表現する測度：State of Charge (SOC)

- 電池残量は充電状態SOC (State of Charge)で表されることが多い。
- SOCは、満充電状態に対する比率として定義され、クーロン、kWh, Ahなどの絶対的な充電量で表せる測度ではないことに注意する必要がある。
- SOCを計測する方法
  - 直接測定法：充放電流を計測。放電流が一定でないことが多い、計測のための放電することが課題。また、電池には内部抵抗があり、充電した分だけ放電できる訳ではない。
  - 重量測定法：鉛酸バッテリに専用の方法。活性化合物の従量変化を測定。電解液である硫酸の比重を計測する。
  - 電圧測定法：電圧から推定する方法。電圧レベル、温度、放電率、電池セルの年齢に依存する。鉛酸バッテリでは電圧と電池残量が比例する傾向があるため有効な測定法であるが、Liイオン電池ではこれが不十分。

21

リチウムイオン電池残量を電圧により計測するのは難しい

- リチウムイオン電池では、放電期間中の電圧降下は小さい。応用にとっては電圧が安定して良いが、残量推定に電圧を使うのは難しい。
- 電圧が急減して残量が空になる。完全放電すると電池寿命が急減する。アプリケーションのための事前警告や、電池寿命を確保するための放電制御が必要。



http://www.mpoweruk.com/soc.htm

23

電池容量は変化し続ける

- 電池容量は、温度と放電測度に依存する
- グラフは、4.2Vを満充電、2.5Vを完全放電電圧とし、電池容量の温度および放電測度に対して計測したもの
- 放電速度が遅ければ-20度でも電池容量はそれなりにあるが、放電速度が速いと-20度では電池容量がほとんど0となることが分かる
- 高速放電で電池残量がなくなっても低速放電に切り換えると放電できることも分かる

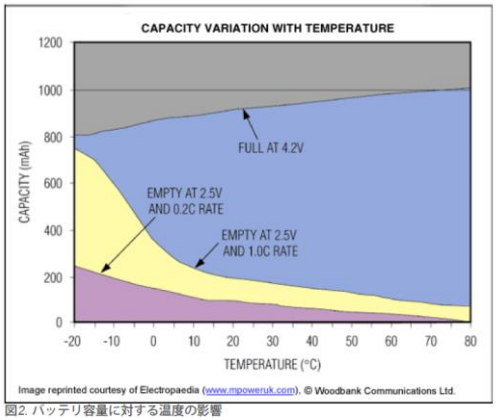


図2. バッテリ容量に対する温度の影響

http://www.mpoweruk.com/soc.htm

25

リチウムイオン電池の自己放電

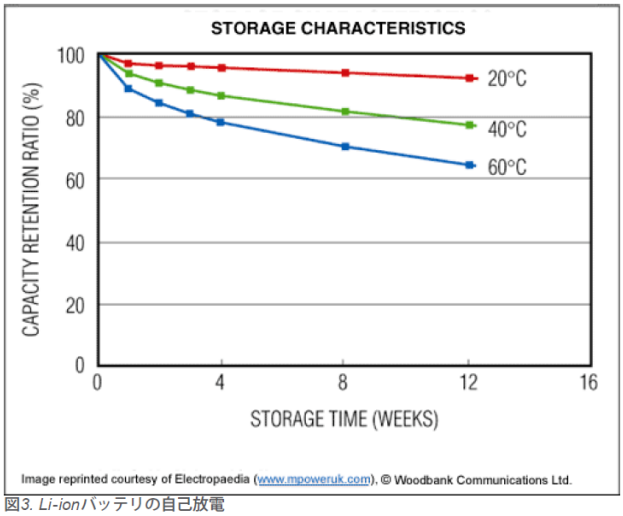


図3. Li-ionバッテリの自己放電

http://japan.maximintegrated.com/app-notes/index.mvp/id/3958 27

リチウムイオン電池容量の経時変化

- バッテリ容量は充放電サイクル数に応じて低下する
- サービスライフ：初期容量の80%まで低下する充放電サイクル数。標準的な値は300～500
- 充放電だけでなく、時間によっても容量が減少する。25℃では1年で20%、40℃では35%減少する可能性がある。部分的に充電された電池の経時劣化は緩やか。

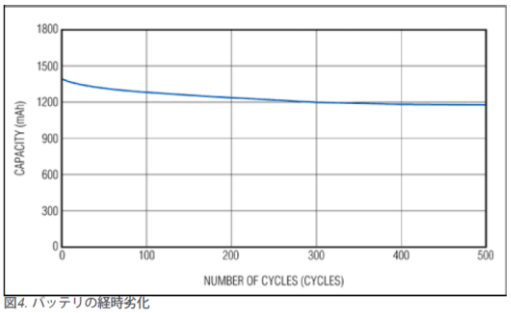


図4. バッテリの経時劣化

http://japan.maximintegrated.com/app-notes/index.mvp/id/3958

29

リチウムイオン電池の容量と充電量の正確な測定

- 電圧による電池残量推定
  - なだらかに電圧降下する特性を活用
  - 精度は高くない
- 正確な推定には、温度、充電サイクル数、電流、製造後期間などの考慮が必要
  - 残量表示の学習プロセスをクーロンカウントと組み合わせて、数パーセント以内の精度を持つ残量表示を実現できる
  - 理論モデルによるSOC推定は実用的でない
  - 実用的な推定法
    - 電池特性評価と、動的測定（放電流や温度、電圧、電流）により推定する
    - 電池特性は、電池セルの性能を温度、放電速度などの関数としてテーブル形式で表現

31

残量表示のためのIC

表2. 残量表示回路の種類

Part	Type of Fuel-Gauge IC	Function in Battery Pack	Function in Host System
DS2762	Coulomb counter	Measurement	Algorithm + display
DS2780	Fuel gauge	Measurement + algorithm	Display
MAX1781	Programmable fuel gauge	Measurement + flexible algorithm	Display

バッテリモニタとも呼ばれるクーロンカウンタは、電荷、温度、電圧、負荷サイクル、および時間を含む、前述したバッテリのパラメータの測定、計数、および変換を行うICです。クーロンカウンタは測定した変数の処理を行わないため、インテリジェント型デバイスではありません。そうしたデバイスの1例であるDS2762には、電流を測定するための極めて正確な25m の抵抗が最初から内蔵されています。温度、バッテリ電圧、および電流の監視を行い、すべての測定値をバッテリパック内またはホストシステム内に存在するマイクロコントローラから読み取ることができるように、1-Wire®バスを備えています。また、Li二次電池にとって最も重要な、必須の安全回路も提供します。結果として、柔軟性が高くコスト効率に優れたシステムが実現しますが、それには相当量の知識と開発工数が必要になります(ただし、そのコストはICベンダが提供するソフトウェア、モデル、およびサポートによって補償されます)。

クーロンカウンタに代わるアプローチの1つを提供するのが、残量表示です。これらのオールインワンデバイスは、学習アルゴリズムを用いた残量計算ルーチンを実行し、それに必要なすべての計測を自力で行います。残量表示は、スマートバッテリと呼ばれる、インテリジェント型の自律的なバッテリに搭載されるのが一般的です。統合型の残量表示によって開発工数が大幅に減少するため、このアプローチは短いタイムトゥマーケットを必要とするアプリケーションに最適です。そうした残量表示の1つであるDS2780では、1-Wireバスを使用したホストによるSOCの読取りが可能になっています。

もう1つの選択肢は、プログラマブル残量表示によって提供されます。これには、大幅な柔軟性を実現するマイクロコントローラが内蔵されています。たとえばMAX1781には、RISCコア、EEPROM、およびRAMが内蔵されています。このデバイスを使用することによって、各種のバッテリモデル、残量計算ルーチン、および測定法を、必要に応じて開発者が実装することができます。内蔵のLEDドライバが、シンプルではあるが正確なSOC表示をサポートします。

33

http://japan.maximintegrated.com/app-notes/index.mvp/id/3958

付録B Dynamic-Help Methodにおける更なる性能向上手法

動的負荷分散により，各 uTuple サーバが担当するハッシュ区間の数は増加する．uTupleSpace に登録される uTuple の数やデータ種別が継続的に変化し，頻繁に動的負荷分散が起きると，それぞれの uTuple サーバがハッシュ空間の全区間を担当することにもなり得る（これが「最悪の状況」である）．最悪の状況においては，全ての EF および CA が全 uTuple サーバに複製されるため，オーバーヘッドは最大に達してしまう．

本付録では，そのような状況になるのを防ぐ 3 つの手法について紹介する．4.4.4.1 では，最悪の状況における評価結果について述べたが，これらの手法により最悪の状況に陥るのを極力避けることができる．

手法 1：分担管理サーバは同じハッシュ区間を既に担当している uTuple サーバを優先的にヘルパーサーバとして選ぶ

Dynamic-Help Method では，一つのハッシュ区間を，蓄積可能フラグ OFF の 0 台以上の uTuple サーバと，蓄積可能フラグ ON の 1 台以上の uTupe サーバが担当する．これらの uTuple サーバのうち，いずれかが過負荷状態となった場合，同ハッシュ区間を担当する別の uTuple サーバの負荷が軽く，ヘルパーサーバとなり得る場合には，分担管理サーバは当該 uTuple サーバを別のハッシュ区間を担当している uTuple サーバよりも優先的にヘルパーサーバとして選出する．これにより，動的負荷分散によって担当ハッシュ区間が増加する uTuple サーバが発生することを抑えることができる．例えば，同ハッシュ区間を担当する別の蓄積可能フラグ ON の uTuple サーバの負荷が十分に低ければ，過負荷となった uTuple サーバは自身の蓄積可能フラグを OFF にするだけでよく，また同ハッシュ区間を担当する蓄積可能フラグ OFF の uTuple サーバの負荷が十分に低ければ，そのフラグを ON にすることで負荷分散が可能である．なお，これらの場合には，前記の動作例と比べて EF，CA の複製が不要となるなどの違いがある．

手法 2：処理負荷の動的負荷分散は徐々に行う

Dynamic-Help Method では，一つの uTuple サーバは複数のハッシュ区間を担当する．uTuple サーバの処理負荷が過剰となり，処理負荷の動的分散を行う際，分担管理サーバはその uTuple サーバが担当する全ハッシュ区間について，ヘルパーサーバを立てる必要は必ずしもない．そのいくつかのハッシュ区間に対応する処理負荷が軽くなれば十分なこともある．無駄にヘルパーサーバを立てることは最悪の状況へ近づくことにつながるため，全ハッシュ区間について一気に動的負荷分散を行うのではなく，一部行い，負荷の減少が十分でなければ，さらにまた一部行う，という方式をとる．これにより，担当するハッシュ区間が増加する uTuple サーバ数を抑えることができる．

手法 3 : 負荷監視サーバは uTuple サーバがハッシュ空間の各区間を担当する必要があるか否かを確認する

蓄積可能フラグが OFF となっている uTuple サーバには，新たに uTuple が蓄積されることはない. また前述のように, 各 uTuple には有効期間が設けられ, 有効期間切れの uTuple はガベージコレクションによって uTuple サーバから削除される．そのため，uTuple サーバに蓄積された uTuple は時間の経過とともに有効期間切れとなって削除され，いずれ全て消えてしまう．こうなると，もはや uTuple サーバ上に EF や CA のマッチング対象はなくなる. つまり, 当該 uTuple サーバは, 当該ハッシュ区間の担当を継続する必要がなくなる．そこで，負荷監視サーバは，uTuple サーバに問い合わせ，uTuple が存在しない担当ハッシュ区間の有無を確認する. そのようなハッシュ区間がある場合には, 負荷監視サーバは当該 uTuple サーバを当該ハッシュ区間の担当から外す．つまり，分担表の当該行から当該 uTuple サーバのアドレスを削除する.