卒業論文 2014 年度 (平成 26 年度)

T-Ring:センサデータの時間的特殊性に着目した多次元データ分散管理システム

指導教員

慶應義塾大学環境情報学部

徳田 英幸

村井 純

楠本 博之

中村 修

高汐 一紀

Rodney D. Van Meter III

植原 啓介

三次 仁

中澤 仁

武田 圭史

慶應義塾大学 環境情報学部

小町芳樹

ysk@ht.sfc.keio.ac.jp

卒業論文要旨 2014 年度 (平成 26 年度)

T-Ring:センサデータの時間的特殊性に着目した多次元データ分散管理システム

近年，センサネットワークは環境モニタリングや，スマートグリッドなど，我々の生活を豊かにするインフラとして整備されつつある．また，センサノードの小型化やセンサを搭載したスマートフォンの普及により，センサ自体やセンサネットワークが，家庭，個人の規模で利用されつつあり，センサネットワークが我々を支える生活基盤と成り得る将来は遠くはない．

本研究は，センサネットワークがより一般的な生活基盤となり，他の組織や家庭などとデータを公開，共有し，センサの具体的な所在を気にすることなく，自由にデータを利用できる世の中を想定する．この環境下で，公開，共有されたセンサデータを公衆センサデータとする．

この公衆センサデータの管理を行うには，センサデータを緯度，経度，センサタイプを含んだ多次元のデータとして扱わなければならないが，従来の多次元データ管理の研究において，センサデータを対象とした研究は数が多くない．また，センサデータを対象にしていても，センサデータの時間的特殊性という性質に着目している研究は存在しない．

本研究は, このセンサデータの時間的特殊性に着目した多次元データ管理システムである T-Ring を提案する．T-Ring は時間を多次元データにおける 1 つの属性とせず，保存先変更の基準として扱うことで，データの保存，取得時の計算量を減らし，処理時間を減少させる．さらに，本研究ではこの T-Ring システムの性能を評価することで，有用性を証明する．

キーワード：

公衆センサデータ, 多次元データ管理，P2P，分散ネットワーク，構造化オーバーレイネットワーク

慶応義塾大学環境情報学部

小町芳樹

Abstract of Bachelor's Thesis Academic Year 2014

T-Ring:Sensor Data Oriented Decentralized Multidimensional Indexing Focusing on Particularity of Time

In recent years, sensor networks are being developed as an infrastructure to enrich our lives such as environmental monitoring and smart grids. In addition, by the spread of sensor equipped smartphones and downsizing of sensor nodes, sensor networks (or the sensor itself) are being used in the scale of family or personal. In the future, sensor networks will become the foundation of our life supporting.

This research assumes that sensor networks could become the foundation and that everyone could get data from anywhere without being concerned about which sensor network they belong to. In this environment, these published and shared sensor data are called "public sensor data" in this paper.

To manage this public sensor data, sensor data should be operated as Multidimensional Indexing which has latitude, longitude, sensor type and so on. However, in general works of Multidimensional Indexing, these researches were hardly intended on sensor data. No reserach focused on particularity of sensor time.

This research proposes T-Ring which is Multidimensional Indexing system for public sensor data. This is the first research focusing on particularity of sensor time. T-Ring doesn't use time attributtion in multidimensional data and use a criterion of deciding the place of storage. T-Ring reduces amount of calculations on storing and retrieving of data and processing speed. This paper evaluates this system and proves feasibility.

Keywords :

Public Sensor Data, Multidimensional Indexing, P2P, Decentralize Network, Structured Overlay Network

Yoshiki Komachi

Faculty of Environment and Information Studies
Keio University

目次

図目次

表目次

第 1 章

序論

本章では，まずセンサデータの増加などの本研究の背景を述べ，次いで，センサデータの公衆一般化において発生する問題の解決という本研究の目的について言及し，最後に本論文の構成を示す.

1.1 本研究の背景

近年センサの低価格化，高性能化により，センサネットワークが普及し，それに伴いセンサ用のオペレーティングシステムの研究も発展を遂げている．センサネットワークのオペレーティングシステムには主に 2 種類あり，省資源性かつ低オーバヘッドを実現したイベントモデルと，リアルタイム処理を可能としたマルチスレッドモデルが存在している．センサネットワーク用のオペレーティングシステムではイベントモデルが主流となっている．

イベント駆動型の TinyOS は，省資源かつ低オーバヘッドの実現に成功しているが，リアルタイム処理を行うことができていない．また，言語も nesC という特殊な言語を使用しているため，プログラムが非常に書きにくいという欠点も存在する．それに対して，Nano-RK はリアルタイム処理をサポートしているが，イベント駆動型のオペレーティングシステムと比較すると資源の消費が大きく，高オーバヘッドである．

Contiki はイベント駆動型のオペレーティングシステムで，使用言語は C 言語である．Contiki は Protothreads と呼ばれるメカニズムを使用しており，イベント駆動型でありながらスレッドの切り替えを行うことができる．そのため，イベント駆動型の特徴となっているプログラムの書きにくさを解消することに成功している．しかし，タスクを処理する優先順位を明確に決める手法が定められていないため，タスクの切り替えはできるが，未だにリアルタイム処理は不可能なのが現状である．

1.2 本研究の目的

Contiki はイベント駆動型でありながら，マルチスレッドでタスクを変更することができるため，リアルタイム処理と親和性があると推測できる．本研究では，Contiki にスケジューラを実装し，省資源性かつ低オーバヘッドでありながら，リアルタイム処理の可能なオペレーティングシステムの実現を目的とする．最後に，本システムの評価を行うとともに，有用性を証明する．

1.3 本論文の構成

本論文では，2 章において，センサネットワークの一般公衆化とアプリケーションの分類から，そのデータ管理において必要な機能要件を示し，センサデータの時間的特殊性について述べる．3 章では，センサデータではない公衆コンテンツの管理と DHT を用いた広域公衆センサデータについての関連研究を紹介する．4 章，5 章では，センサデータの時間的特殊性に着目したデータ管理システムを示す．6 章で，提案手法に対する評価を行い，7 章で本研究のまとめを行う．

第 2 章

無線センサネットワーク

本章では，最初にセンサネットワークの一般公衆化を説明し，公衆センサデータ管理における必須な機能要件である地理的探索について述べる．次に，公衆センサデータ管理のシステム設計の際の重要な概念であるデータ管理の時間的密度を解説し，最後にセンサデータの時間的特殊性とそれに起因する公衆センサデータ管理の問題点を述べる．

2.1 はじめに

2.2 無線センサネットワークにおけるオペレーティングシステム

無線センサネットワークのオペレーティングシステムには主に 2 種類あり、イベントモデルとスレッドモデルが存在している。無線センサネットワーク用のオペレーティングシステムではイベントモデルが主流となっている。

2.2.1 イベントモデル

イベントモデルで構築されたオペレーティングシステムは全てのタスクをイベントによって起動し，run-to-completion で実行する形態のオペレーティングシステムである．イベントモデルはひとつのイベントループと多数のイベントハンドラから構成される．イベントループはイベントの到着を待ち，イベントが届くとイベントに関連付けられているイベントハンドラを実行する．イベントモデルではイベント駆動型プログラミングによってアプリケーションが記述される．イベントハンドラは寿命の短い run-to-completion で記述され，プリエンプションされることがない．つまり，イベントモデルはタスクは関数呼び出しと等価であり，実行ストリームがひとつで実現されるため各タスクでローカル変数の領域を共有可能なので省資源かつ低オーバヘッドで並列性を実現できる．また，各タスクが不可分に実行されるので共有資源に対する排他制御が不要となり，安全性が高い．さらに，CPU の特殊な機能を用いなくても実装できるので移植性も高い．

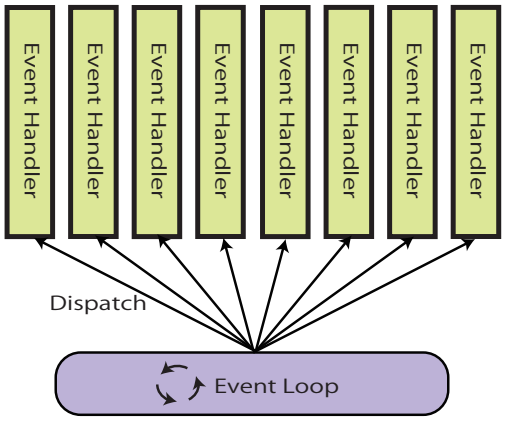


図 2.1　イベントモデル

2.2.2 スレッドモデル

スレッドモデルは複数のスレッドから構成される．各スレッドはそれぞれ独立に実行ストリームを持っており，低い優先度のスレッドは高い優先度のスレッドにプリエンプションされるという特

徴を持つ．スレッドモデルではユーザはあたかも CPU を占有しているかのように一連の処理をひとつのスレッドとして記述することができるのでプログラムが書きやすい．また，プリエンプションを行うことも想定しているのでハードリアルタイム処理をサポートすることができる．

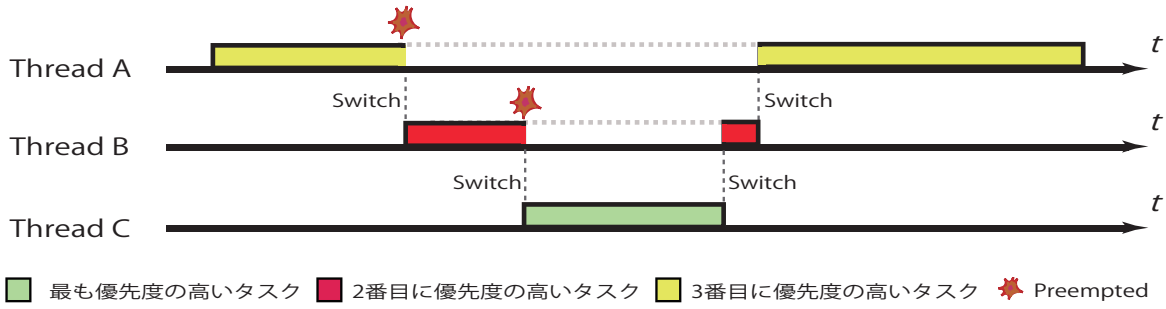


図 2.2　スレッドモデル

ジョブの実行を設定された時間通りに作動させることをリアルタイム処理という．リアルタイム処理には主にハードリアルタイム処理とソフトリアルタイム処理，の 2 種類がある．

2.2.2.1　ハードリアルタイム処理

課せられた処理が期限内に終了しなかったとき，システム全体に致命的なダメージが生じてしまうリアルタイム処理のことをハードリアルタイム処理という．したがって，期限内での終了が保証されなければならないシステムに用いられる．

2.2.2.2　ソフトリアルタイム処理

ソフトリアルタイム処理を行うシステムでは，期限内に処理が終了しなくてもシステム全体に致命的なダメージを与えることはない．ただし，処理自体の価値は終了期限とともに減少していく．

2.3　本研究の問題意識

しかしながら，ユーザが一連の処理を細かい処理に分割しなければならないのでプログラムが書き辛いという問題が発生する．

さらに，イベントモデルではタスクのプリエンプションをしないことを前提に設計されているのでハードリアルタイム処理のサポートができない．しかしながら，プリエンプション時のオーバヘッドの大きいことや必要とされる資源が多いこと，スレッド間の共有資源へのアクセス制御が必要となるために安全性が損なわれるなどの問題を持っている．

表 2.1　オペレーティングシステムの比較

	メリット		デメリット	
TinyOS	省資源	低オーバヘッド	リアルタイム性の非サポート	プログラムが記述しにくい
Nano-RK	リアルタイム性のサポート		資源の消費が大きい	高オーバヘッド

2.4　まとめ

本章では，まず，センサネットワークの一般公衆化について述べた．次いで，公衆センサデータがを扱ったシステム設計をする際に，データ管理の時間的密度という要素を考慮すべきであることを示した．そして，センサデータが時間的特殊性を持った多次元データであることを述べ，また，それに起因するセンサデータ管理における問題を提起した．

第 3 章

関連研究

本節では，最初に多次元データ管理 Multidimensional Indexing（MI）の分野における関連研究を挙げる．次に，Contents Delivery Network（CDN）におけるコンテンツの特徴を利用したレプリケーションを行った研究を挙げる．最後に，センサデータを多次元データとして扱った研究を挙げる．

3.1 多次元データ管理

1990 年代から，データベースの分野において，多次元のデータをどのように管理するべきか Multidimensional Indexing（MI）[1] が議論されている．そして，地理学，ロボット工学，環境保護学などの様々な分野でこの MI は応用されている．1990 年代の MI の中心はデータが集中管理されている環境を想定したアルゴリズムであり，VA_File[2]，Hilbert R-tree[3]，GHT[4] などがその代表である．2000 年代からは，P2P を用いた分散型の MI が主流となる．2000 年代前半に提案された Chord[5]，CAN[6]，SkipGraph[7] が代表である．2000 年代後半から現在にいたるまでにこれ以外の数多くのアルゴリズムが提案されているが，大半がこのいずれかのアルゴリズムの変形であって，大きなアルゴリズムの変化はない．

センサデータが公衆化し，広域に管理される場合，2.2 節で述べられているように，データの値以外のインデックスが付与される必要がある．よって，公衆広域センサデータを多次元データとして管理しなければならない．

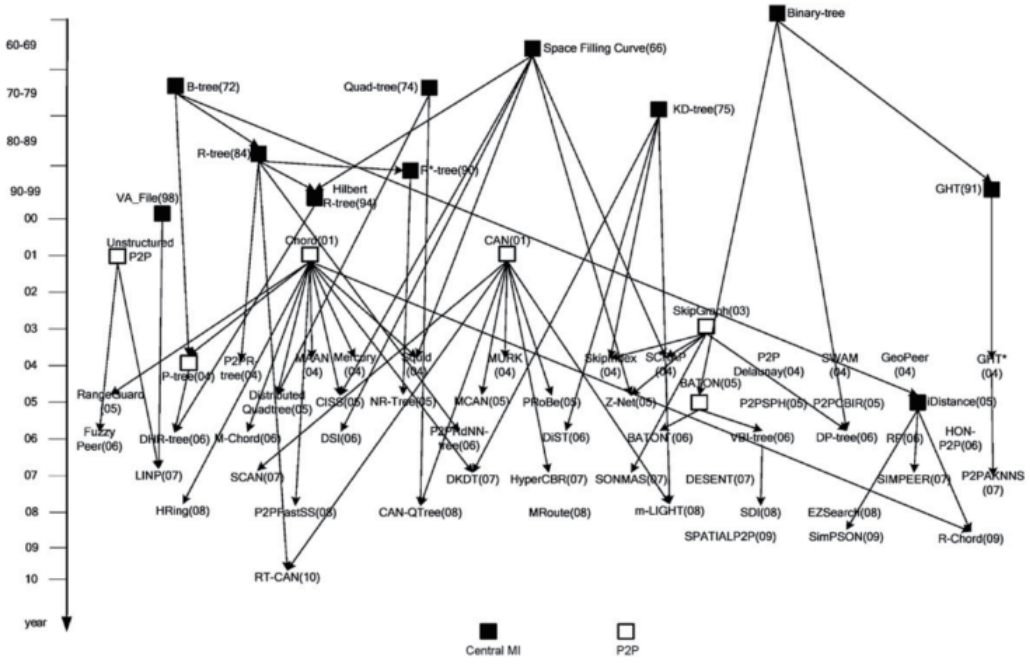


図 3.1 Multidimensional Indexing の歴史（参考：P2P-based multidimensional indexing methods. Journal of Systems and Software 2011[8]）

3.2 文書や音楽などのコンテンツの広域管理

この多次元データ管理は，Akamai[9] などに代表される，Contents Delivery Network（CDN）などで利用されている．CDN は世界中の数十，数百のデータセンターで P2P ネットワークを構

築し，ユーザーに音楽や動画などを提供している．コンテンツには，サイズ，言語などの多次元のインデックスが付与されている．CDN は，コンテンツの特徴を抽出し，レプリケーション先を決定し，効率的なコンテンツの配送を実現している．Cha ら [10] は，Flickr[11] のコンテンツがどのように伝播するのかの調査をしている．Scellato ら [12] は twitter[13] や facebook[14] などの SNS などに載せられたコンテンツとその発言などのいち情報から，コンテンツの地域性を割り出し，レプリケーションを効率的に行う手法を提案している．Akamai では，トラフィックの状況などを公開しており，時間帯や地域による傾向を捉えることができる．関連研究ではコンテンツの言語から地域性を抽出し，対象地域のデータセンターに予めコンテンツをレプリケーションすることにより，効率化を実現している．この他にも，コンテンツや利用状況の特徴を抽出するために [15][16][17][18] などのインターネット計測の分野の研究結果も利用されている．

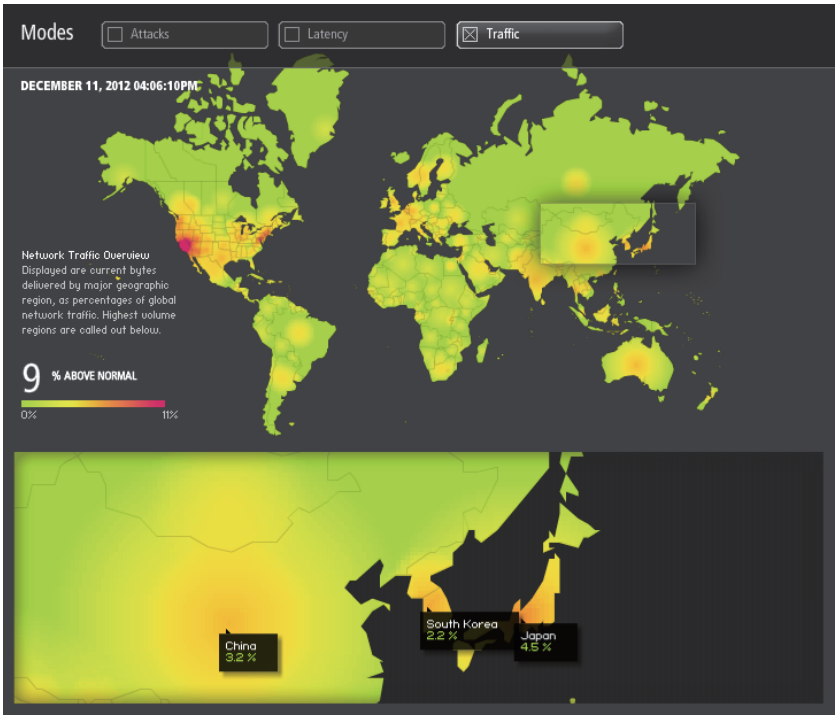


図 3.2　Akamai によるトラフィック情報の公開（参考：Akamai[9]）

センサデータは 2.3 節で取り上げた，音楽や動画とは異なる時間的特殊性に加え，言語などから由来する地域性も存在しないため，コンテンツの特徴からレプリケーションの場所を決定するという手法を取ることができない．

3.3　DHT を用いた公衆センサデータ管理

広域センサデータを多次元データとして，MI のアルゴリズムを用いた研究に Synapse[19] がある．この研究では，従来の，発せられたセンサデータを地理的に近い保存ストレージに保存すると

いう手法では，イベントの発生による突発的な人口過密が発生した際に，特定のセンサデータ保存ストレージに対する保存，取得のクエリが集中すると主張している．そこで，センサデータの緯度，経度，センサタイプ，データの時間を空間充填曲線 [20] 図 3.3 を用いて 1 次元化する．空間充填曲線は Lawder らが提案した，DHT アルゴリズムにおいて，多次元データを 1 次元化する手法で，これにより多次元データをハッシュ関数にかけることが可能になる．この手法は MI における基本的な手法であり，2009 年に Kantere らが提案した SPACIALP2P[21] という MI アルゴリズムでも用いられている．このハッシュ化した値を元に DHT ネットワークを構築することで，特定の保存ストレージに対するクエリの集中を防いでいる．

しかし，Synapse ではセンサデータの時間的特殊性が考慮されていないため，利用状況によっては，パフォーマンスが低下してしまう可能性がある．

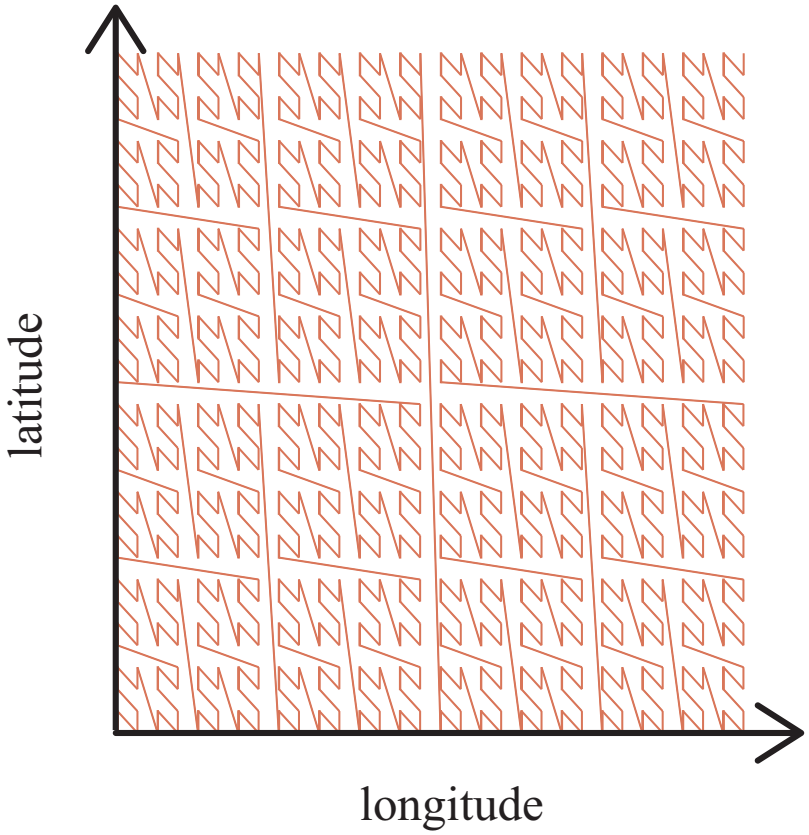


図 3.3　空間充填曲線:Z-order

3.4　まとめ

本章では，まず，公衆広域センサデータが多次元データであるということから，本研究の根幹を成す，Multidimensional Indexing（MI）という研究分野を紹介した．次に，Contents Delivery Network（CDN）という分散したデータセンターで多次元データを管理するネットワークという

観点から捉え，センサデータには CDN で扱われるような特殊性が存在しないことを述べた．最後に，公衆広域センサデータの分散管理手法を紹介し，センサデータの時間的特殊性が考慮されていないことに言及した．

第 4 章

Contiki - a Lightweight and Flexible Operating System for Tiny Networked Sensors

本章では，最初に，本研究が提案する T-Ring という公衆センサデータ管理システムにおける，時間概念の扱いと，それに伴った保存アルゴリズムの概念説明を行う．次に，集中管理と Synapse とのデータ管理の時間的密度と計算コストの比較を行う．詳細な設計については次章で解説する．

4.1 Protothreads

既に述べたように，イベント駆動型は組み込みシステムやセンサネットワークに対して，主流なオペレーティングシステムとなっている．しかし，イベント駆動型はメモリのオーバヘッド低く維持できる一方で，ユーザが一連の処理を細かい処理に分割しなければならないため，プログラムが書き辛いという問題が発生する．Protothreads はイベント駆動型プログラムをマルチスレッド型のように記述することができるため，メモリのオーバヘッドを抑えつつ，イベント駆動型の欠点を補うことが可能となる．本節では，Protothreads の機能について詳しく述べる．

4.1.1 メモリ

マルチスレッド型のオペレーティングシステムでは，それぞれのスレッドにそれぞれのスタックを必要とする．しかし，メモリが制限されているセンサネットワークのようなシステムでは，スタック用のメモリは静的に保持されなければならないため，このメモリを他の目的で使用することはできない．

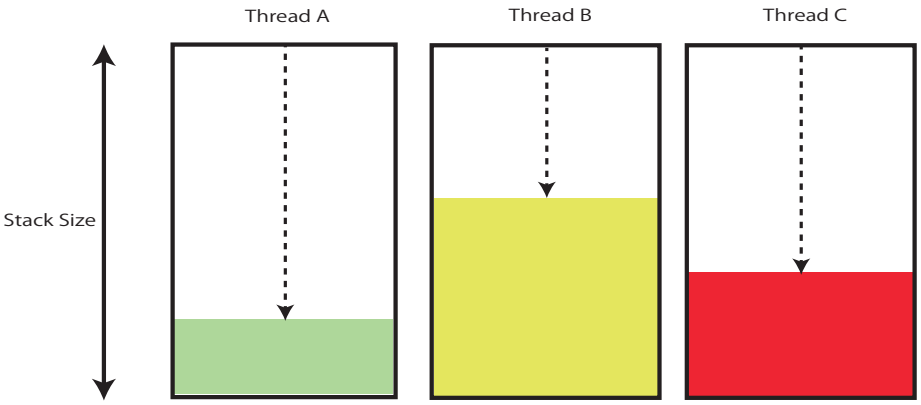


図 4.1　一般的なスレッドモデルにおけるスタック

それに対して，Protothreads を使用した場合，すべてのプログラムは同じスタックを共有し，実行されることとなる．つまり，Protothreads を利用しているオペレーティングシステムではそのような事態を防ぐために，マルチスレッドを実現しつつも，ひとつのスタックをあたかも複数個あるように見せかけている．

4.1.2 タスクの切り替え

一般的なマルチスレッド型のオペレーティングシステムは，タスクの割り込みがあった際には，レジスタの状態を保存することで変数の値を保持し，割り込まれたタスクの実行が終了したときに，再びレジスタから変数を読み込み，処理を途中から再開する．

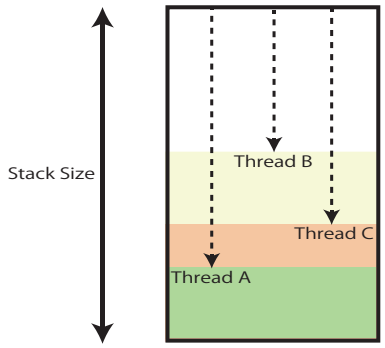


図 4.2　Protothreads におけるスタック

しかし，Protothreads を使用した場合，状態をレジスタに保存することはせずに，戻り値を利用することでタスクの CPU の利用を放棄する．さらに，アプリケーション内の変数は静的な変数を採用しているため，低オーバヘッドを実現しつつ，一貫性を保っている．

戻り値をもとにしたタスクの切り替えを行う際には，スケジューラに対してプログラムのファイルの行数を return し，各タスクが CPU 利用権限を再度取得したとき，行数を条件分岐し，前回中断した箇所から実行を再開することとなる．

ただし，現在実行中のタスクよりも優先度の高いタスクが実行待ちになった際に，他のマルチスレッドのオペレーティングシステムで実現されているように，ループ内の処理を実行しているタスクに割り込みをし，タスクを切り替えて実行することはできない．

現在では，タイマー割り込みを目的とした利用をされていない．タイマーと現在時刻を比較し，発火時刻を過ぎている場合，タスクを実行待ちのキューへと加える．タイマーを実行するタスクは他のタスクと同じ優先度で周期的に実行される．

4.2　イベント

イベントモデルである Contiki では、イベントが発生するとプロセスが実行される。本節では Contiki で扱われる非同期イベントと同期イベントの違いについて説明する。

4.2.1　非同期イベント

非同期イベントが発生すると、そのイベントはカーネルのイベントキューに挿入され、実行時に First In First Out で呼び出される。

Asynchronous events are delivered to the receiving process some time after they have been posted. Between their posting and their delivery, the asynchronous events are held on an event queue inside the Contiki kernel.

The events on the event queue are delivered to the receiving process by the kernel. The kernel loops through the event queue and delivers the events to the processes on the queue by

invoking the processes.

The receiver of an asynchronous event can either be a specific process, or all running processes. When the receiver is a specific process, the kernel invokes this process to deliver the event. When the receiver of an event is set to be all processes in the system, the kernel sequentially delivers the same event to all processes, one after another.

Asynchronous events are posted with the process_post() function. The internals of the process_post() function is simple. It first checks the size of the current event queue to determine if there is room for the event on the queue. If not, the function returns an error. If there is room for the event on the queue, the function inserts the event at the end of the event queue and returns.

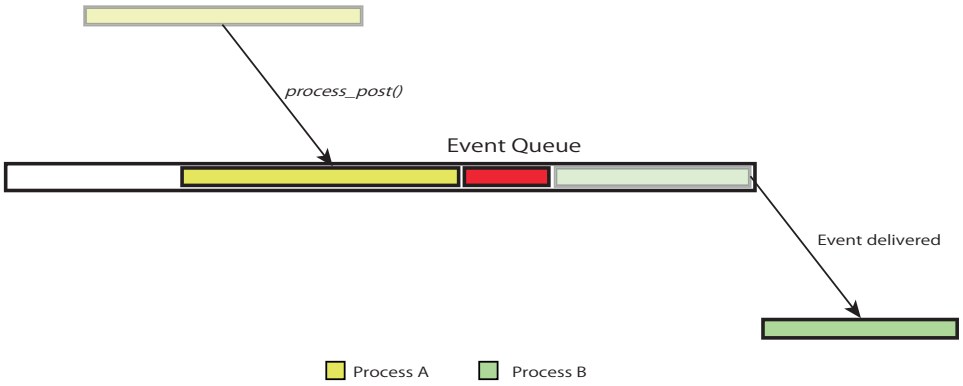


図 4.3　非同期イベントの実行

4.2.2　同期イベント

非同期イベントに対して、同期イベントでは

Unlike asynchronous events, synchronous events are delivered directly when they are posted. Synchronous events can only be posted to a specific processes.

Because synchronous events are delivered immediately, posting a synchronous event is functionally equivalent to a function call: the process to which the event is delivered is directly invoked, and the process that posted the event is blocked until the receiving process has finished processing the event. The receiving process is, however, not informed whether the event was posted synchronously or asynchronously.

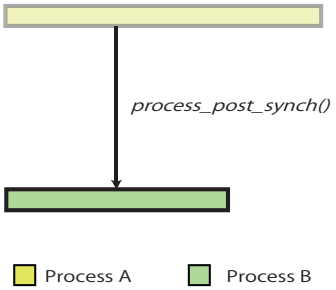


図 4.4　同期イベントの実行

4.3　ポーリング

4.4　イベントタイマー

Contiki オペレーティングシステムには

4.5　まとめ

本章では，まず，保存ピアの計算量を削減し，Synapse よりもデータ管理の時間的密度が高いアルゴリズムを提案することを述べた．次に，T-Ring において時間という属性を DHT における保存ピアの決定における一つ要素としてではなく，保存ピアの変更を行うための要素として扱うことを述べた．そして，時間的密度，計算コストの観点から集中管理，Synapse，T-Ring の 3 つの手法の比較を行った．また，センサデータの時間的特殊性に着目した多次元センサデータ分散管理システムである T-Ring の詳細な設計概念，設計実装について述べた．T-Ring は P2P のトポロジとして，代表的なアルゴリズムである Chord に準拠した 1D トーラスを用いている．保存ピアの参加や離脱なども同様に Chord に準拠している．また，多次元データを扱うにあたり，Z-order により 1 次元化に言及した．時間的特殊性については，chunk と SP という 2 つの時間概念を導入し，保存ピア変更の基準とした．また，ユーザが T-Ring システムを用いて，データ取得のクエリを送る際の，形式を定義した．

第 5 章

T-Ring の設計実装

本章では，T-Ring における詳細な設計，実装について解説する．

5.1　システム構成

本節では，T-Ring システムの設計について述べる．T-Ring を構成する保存ピアはネットワークモジュール，センサ情報管理モジュール，データ保存モジュール，データ取得モジュール，保存ピア発見モジュール，センサ管理モジュール，アプリケーションモジュールの 7 つのモジュールによって構成される．図 5.1 がシステム構成図であり，各モジュール間でのデータの受け渡しが記述されている．

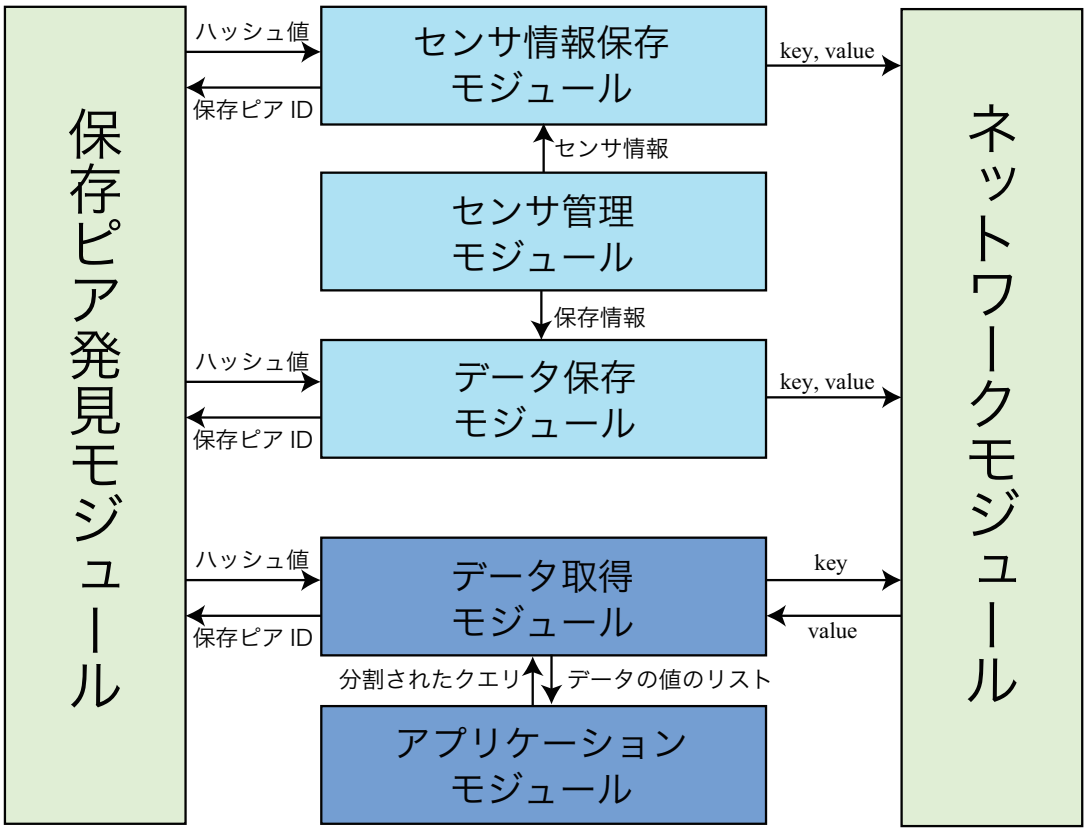


図 5.1　システム構成図

5.1.1　ネットワークモジュール

ネットワークモジュールは保存ピア同士で協調し，1D トーラス型のオーバーレイネットワークを構築及びデータに関する通信を行うモジュールである．本モジュールでは，他の保存ピアのデータ取得モジュールから，データ取得のクエリが送られた場合に，対象のデータをクエリ送信元の保存ピアに送信する．

本モジュールの実装はオープンソースである OpenChord の一部を参考にしている．保存ピアの

Join，Retrieve などの保存ピアによるネットワークの構築の部分において利用している．

5.1.2 センサ情報管理モジュール

本モジュールは，センサの緯度や経度など，センサデータの値以外の多次元データとしてのセンサ情報を適切な場所に保存することで，データの取得を可能にするモジュールである．

センサデータの取得を考えた際に，一般的な手法は，センサノードのアドレスを指定して取得するが，T-Ring が想定している環境では，センサデータをアドレス以外の他の要素から取得する．よって，取得を行う時には，取得されるセンサデータの情報が必要である．このために，T-Ring では，対象のセンサノードの緯度，経度，センサタイプ，マスターピア=0 で一次元化，ハッシュ化した値を担当する保存ピアに対して，このセンサ情報を登録する．登録はセンサデータの値の保存と同様に行う．センサデータの保存については，次節で詳細に述べる．保存におけるキーとして，緯度，経度，センサタイプ，マスターピア=0 の 4 次元値を 1 次元化，ハッシュ化した値を，バリューとして，そのセンサの chunk と SP の時間のセットを保存する．データの取得については，5.4 で詳細に述べるが，データの取得の際には，緯度，経度，センサタイプ，chunk，SP の時間を必要とする．緯度，経度，センサタイプ，マスターピア=0 の担当ピアに取得に必要なセンサ情報を登録することによって，chunk と SP の時間を知ることが可能になる．

本モジュールでは，最初に，センサに関する情報が登録されている SensorInfo.csv から，センサ ID をキーとして，センサデータの保存に必要である，緯度，経度，センサタイプ，chunk，SP の時間を取得する．取得された情報から，緯度，経度，センサタイプ，マスターピア=0 の 4 次元値を Z-order 関数によって処理し，返されたバイナリ値を保存ピア発見モジュールに渡す．保存ピア発見モジュールは，担当の保存ピアの保存ピア ID を返してくるので，そのピアに対して必要な情報を登録する．

5.1.3 データ保存モジュール

本モジュールはセンサデータを保存する保存するモジュールである．

T-Ring では，センサデータの時間に従って保存先を決めるため，センサ管理モジュールから，センサ ID，対象データの緯度，経度，センサタイプ，chunk，SP の時間，データの時間，センサの設定時間を取得する．本モジュールでは，フィールドとして，センサ ID をキーに，次に保存すべき保存ピアの ID をバリューとする Map を所持している．送られてきたセンサ ID と Map 内のセンサ ID を比較し，登録されていない場合は，データの時間，センサの設定時間，chunk から，まず，マスターピアの番号を決定する．次いで，そのマスターピアから Successor を辿る回数を計算する．そして，緯度，経度，センサタイプ，マスターピアの番号の 4 次元からハッシュ化をし，その値と Successor を辿る回数を保存ピア発見モジュールに送る．その結果として保存ピア ID を受け取る．データが保存されるセンサの ID と Map 内のセンサ ID が一致していた場合は，その Map のセンサ ID に対応する保存ピアの ID を取得する．次に，その ID の保存ピアに対して，キーを緯

度，経度，センサタイプの 1 次元化とハッシュ値，バリューをデータの値として保存する．

また，保存が行われると，次に保存するべき保存ピアの計算が行われる．ここからはセンサの ID とデータの時間だけを監視する．この計算を行うために，センサ ID をキーとして，最新のセンサデータが保存された保存ピアの ID，マスターピアの番号それぞれをバリューとして保存する 2 つのフィールドが存在する．次に送られてきたデータが前回のデータと比較して chunk が異なっていた場合，保存ピア発見モジュールに対して，最新の保存データが保存された保存ピア ID と Successor を辿る回数が送られる．これに対して，対象の保存ピアの ID が返される．この保存ピアの ID を次に保存すべき保存ピア ID として更新する．次に，chunk が同一であった場合，最新の保存データが保存された保存ピア ID を次に保存すべき保存ピア ID をとして更新する．マスターピアが異なっていた場合，センサ ID が登録されていない時と同様の処理を行い，保存ピア発見モジュールから返された保存ピア ID を次に保存すべき保存ピア ID として更新する．

これらの処理が繰り返えされることにより，データの保存が行われる．以上に述べた処理をフローチャートで示す．

5.1.4 データ取得モジュール

アプリケーションモジュールから送られるユーザからの取得のクエリは，範囲を持ったクエリであるので，本モジュールにより，クエリの緯度，経度，半径から分割を行う．これについては図 5.2 が例と図解である．そして，分割を行ったそれぞれの位置について，緯度，経度，センサタイプとクエリで指定された取得時間から，データ保存と同じ手順により，緯度，経度，センサタイプ，マスターピアの番号の 4 次元によるハッシュ値，Successor を辿る回数を保存ピア発見モジュールに送り，その結果として受け取った保存ピア ID に対して，緯度，経度，センサタイプで生成されるキーからバリューを受け取る．この受け取ったバリューのセットを，指定された方式に変換しアプリケーションモジュールに送り返す．指定された方式に関しては，アプリケーションモジュールの解説の際に説明する．

5.1.5 保存ピア発見モジュール

本モジュールは，センサ情報管理モジュール，データ保存モジュール，データ取得モジュールから受け取ったハッシュ値や Successor を辿る回数から，対象の保存ピアを発見し，そのアドレスを各モジュールに送り返す．

T-Ring では 2 種類の保存ピアの発見の方法が存在する，1 つ目は，Finger Table を用いた発見である．Finger Table の計算回数は $O(\log 2N)$ である．この発見はデータ保存モジュール，データ取得モジュールから送られた緯度，経度，センサタイプ，マスターピアの番号の 4 次元によるハッシュ値が前データと異なる場合に，マスターピアを探索するために行われる．また，Synapse における保存，取得では，センサデータ毎にこの計算がなされる．

2 つ目の方法は，データ保存モジュール，データ取得モジュールから送られた緯度，経度，セン

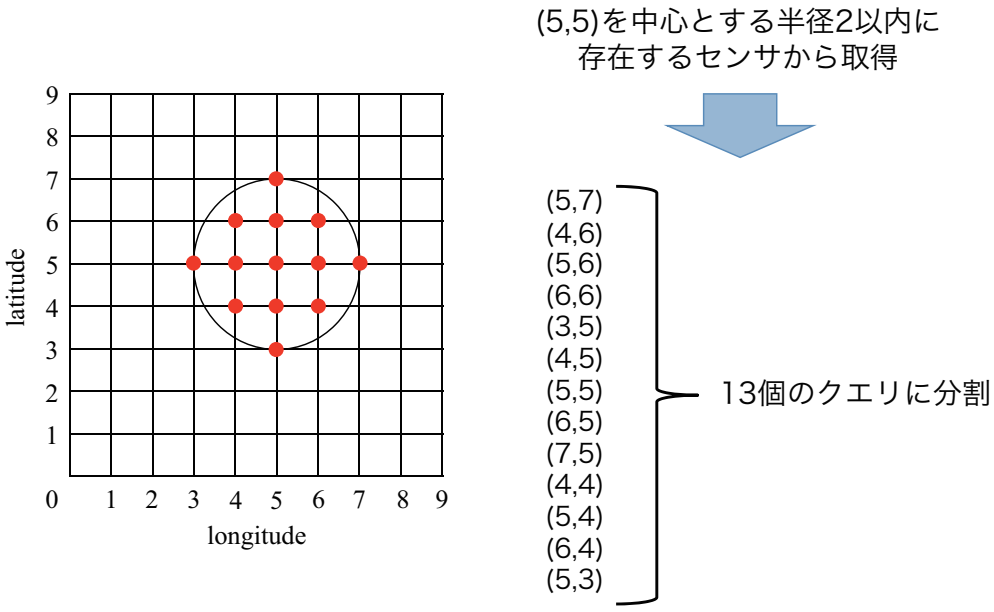


図 5.2　クエリの分割

サタイプ，マスターピアの番号の 4 次元によるハッシュ値が前データと同様であった場合に行われる．この場合，前データで取得したアドレスの Successor のアドレスを取得する．Finger Table による計算は前データの際に行ったため，計算回数は $O(1)$ である．

これらの方法により発見されたアドレスをデータ保存モジュール，データ取得モジュールに送り返す．

Listing 5.1　データ保存時の保存ピア発見

```
public class NextStorePeer {
  private TRingImpl tring;
  private HashMap<String, Node> nextNode;
  private HashMap<String, Long> restTime;
  private HashMap<String, Long> prevSP;
  private HashMap<String, Long> prevTime;
  ArrayList<Long> positions;

  public NextStorePeer(JoinData joinData) {
    this.positions = new ArrayList<Long>();
    this.nextNode = new HashMap<String, Node>();
    this.restTime = new HashMap<String, Long>();
    this.prevSP = new HashMap<String, Long>();
    this.prevTime = new HashMap<String, Long>();
    this.tring = joinData.tring;
  }

  public synchronized Node getNextStorePeer(Node node, DataForStore data,
      Key key) throws CommunicationException {
    setNextStorePeer(node, data, key);
    return nextNode.get(Arrays.toString(key.getBytes()));
  }

  private synchronized void setNextStorePeer(Node currentPeer,
      DataForStore data, Key key) throws CommunicationException {
    long SP = (data.time - data.configTime) / data.startPoint;

    if (nextNode.get(Arrays.toString(key.getBytes())) == null) {
```

```
      this.nextNode
          .put(Arrays.toString(key.getBytes()), currentPeer
              .findSuccessor(currentPeer.getNodeAddingOneID()));
      this.prevSP.put(Arrays.toString(key.getBytes()), SP);
      this.restTime.put(Arrays.toString(key.getBytes()), data.startPoint
          - data.chunk);
    } else {
      if (this.prevSP.get(Arrays.toString(key.getBytes())) != SP) {
        this.positions.add(data.latitude);
        positions.add(data.longitude);
        positions.add(data.type);
        this.positions.add(SP);
        ZorderInterface zorder = new Zorder(positions);
        byte[] SPKey = zorder.getZorder();
        Key masterPeerKey = new ByteArrayKey(SPKey);
        nextNode.put(Arrays.toString(key.getBytes()),
            tring.getResponsiblePeer(masterPeerKey));
        this.prevSP.put(Arrays.toString(key.getBytes()), SP);
      } else if (restTime.get(Arrays.toString(key.getBytes()))
          - data.chunk > 0) {
        this.nextNode.put(Arrays.toString(key.getBytes()), currentPeer
            .findSuccessor(currentPeer.getNodeAddingOneID()));
      }
      this.restTime.put(Arrays.toString(key.getBytes()), data.startPoint
          - data.chunk);
    }
  }
}
```

Listing 5.2　データ取得時の保存ピア発見

```
public class NextRetrievePeer {
  private TRingImpl tring;
  private HashMap<String, Node> nextNode;
  private HashMap<String, Long> restTime;
  private HashMap<String, Long> prevSP;
  ArrayList<Long> retrievedData;
  ArrayList<Long> positions;

  public NextRetrievePeer(TRingImpl tring) {
    this.positions = new ArrayList<Long>();
    this.nextNode = new HashMap<String, Node>();
    this.restTime = new HashMap<String, Long>();
    this.prevSP = new HashMap<String, Long>();
    this.retrievedData = new ArrayList<Long>();
    this.tring = tring;
  }

  public synchronized ArrayList<Long> getNextRetrievePeer(
    DataForRetrieve data, long latitude, long longitude,
    HashMap<String, Long> sensorInfo, Key key)
    setNextRetrievePeer(data, latitude, longitude, sensorInfo, key);
    return this.retrievedData;
  }

  private synchronized void setNextRetrievePeer(DataForRetrieve data,
      long latitude, long longitude, HashMap<String, Long> sensorInfo,
      Key key) throws CommunicationException {
    String tag = Arrays.toString(key.getBytes());
    long currentTime = data.timeFrom;
    long SPTime = sensorInfo.get("SP");
    long chunk = sensorInfo.get("chunk");
    long configTime = sensorInfo.get("configTime");
    long SP = (currentTime - configTime) / SPTime;
    long pastTime = currentTime - configTime - (SP * SPTime);
    long leftChangeTime = SPTime - pastTime;
    long leftTime = data.timeTo - currentTime;

    ArrayList<Long> positions = new ArrayList<Long>();
    positions.add(latitude);
```

```
    positions.add(longitude);
    positions.add(data.type);
    positions.add(SP);
    ZorderInterface zorder = new Zorder(positions);
    byte[] SPKey = zorder.getZorder();
    Key masterPeerKey = new ByteArrayKey(SPKey);

    nextNode.put(tag, tring.getResponsiblePeer(masterPeerKey));

    Node currentNode = nextNode.get(tag);

    while (leftTime > 0) {
      leftTime -= chunk;
      leftChangeTime -= chunk;

      if (leftChangeTime > 0) {
        this.retrievedData.addAll(tring.retrieveData(key,
            this.nextNode.get(tag)));
        this.nextNode.put(
            tag,
            this.nextNode.get(tag).findSuccessor(
        this.nextNode.get(tag).getNodeAddingOneID()));

      } else {
        leftChangeTime += SPTime;
        positions.set(3, positions.get(3) + 1);
        zorder = new Zorder(positions);
        SPKey = zorder.getZorder();
        masterPeerKey = newByteArrayKey(SPKey);
        this.nextNode.put(tag,this.nextNode.get(tag).findSuccessor(this.nextNode.get(tag).
            getNodeAddingOneID()));
        this.retrievedData.addAll(tring.retrieveData(key,this.nextNode.get(tag)));

      }
    }
  }
}
```

5.1.6 センサ管理モジュール

本モジュールはセンサネットワーク内のセンサ毎の緯度，経度，センサタイプ，などを管理を行うモジュールである．センサが環境に設置された際に，そのセンサの情報を所属する保存ピアへ登録する．また，センサの移動により，緯度，経度の変更が発生した際に，その情報を更新する．

センサが環境に設置を行う際，以下のようなセンサ情報登録クラスのインスタンスを用いて SensorInfo.csv に登録を行う．SensorInfo.csv に登録されるエントリは，各センサごとに，sensorId, latitude, longitude, sensorType, chunkTime, startPointTime である．センサの設置場所が変更された際は，sensorId から対象のエントリを発見し，変更を行う．この SensorInfo.csv の情報は，センサ情報保存モジュール，データ保存モジュールで利用される．以下がセンサ情報の登録，変更のメインクラスである．

5.1.7 アプリケーションモジュール

本モジュールはユーザアプリケーションに対してのデータ取得のインターフェースを提供し，ユーザアプリケーションは 5.1.1 で述べたクエリを本モジュールに送信する．本モジュールとは TCP で通信を行い．7777 番ポートを利用する．ユーザアプリケーションとのセッションが確立

し，クエリの受信が完了すると，データ取得モジュールにこのクエリをフォワードし，その結果をユーザアプリケーションが指定した方式で送り返す．この方式の指定には 4 つの方法があり，指定領域内の全データのリスト，最大値，最小値，平均値である．

5.2 まとめ

本章では，まず，ソフトウェアの構成について述べた，そして，個々のモジュールの細かな設計と実装について述べた．特に重要なアルゴリズムについては，各小節の末尾に実際のプログラムを掲載した．

第 6 章

評価

本章では，T-Ring システムの評価を行う．

6.1 評価方針

本研究では，センサデータを多次元データとして扱う際に，時間を他の属性と同様に扱っている Synapse と，センサデータの時間的特殊性を考慮した T-Ring の取得時間の比較を行う．Synapse の手法では，保存，取得の際に，データごとに Finger Table による保存ピアの探索を行わなければならない．よって，本評価では，ネットワークに参加するピア数を固定し，データの保存，取得に共通する，保存ピア探索の計算コストを計測する．次に，保存時，取得時に分けて探索に要する時間の計測を行う．

6.2 評価環境

本研究は，評価環境として，慶応義塾大学湘南藤沢キャンパス内の特別教室内に設置されている 124 台の iMac の内，任意の複数マシンを用いて行う．詳細なスペックなどについては以下の表 6.1 で示す．

表 6.1　実験環境

ホスト名	zmac000-159
本体	iMac 21.5 インチ
CPU	3.06 GHz Intel Core 2 Duo
メモリ	4 GB (2 GB X 2)
ハードディスク	499.76 GB
OS	Mac OS X 10.6.5

6.2.1 実験環境内におけるネットワークレイテンシ

本研究では，データの保存や探索に要する時間を計測するため，各保存ピア間の通信におけるレイテンシが重要になる．そこで，本実験環境におけるマシン間のレイテンシが実験にどの程度の影響を与えるのかを調査するため，事前実験として，各リージョン間の RTT を計測した．各リージョンがどこに存在するかは，図 6.1 において示す．それぞれ ICMP における Echo Message を送信し，リプライが返されるまでの時間の計測を 100 回行い，その平均値，最小値，最大値，標準偏差を表 6.2，6.3，6.4，6.5 で示す．本実験において，実験環境下では RTT が 0.400ms 以下であるという知見が得られた．本システムが用いられる実環境における RTT は 0.400ms より十分に大きいため，本評価では，実験環境内におけるネットワークレイテンシを無視する．

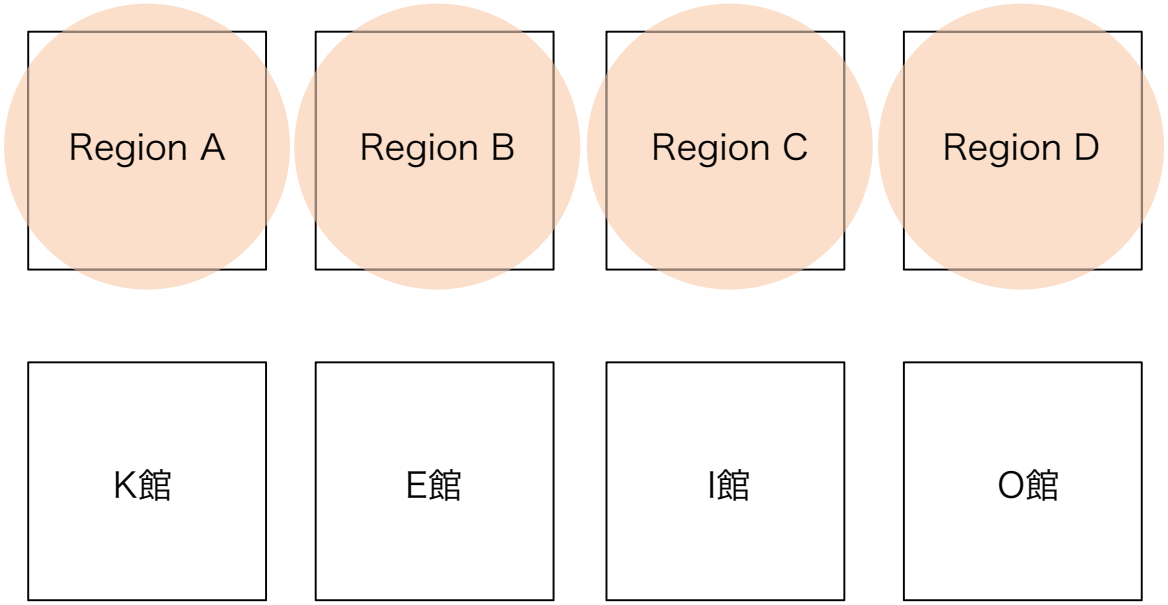


図 6.1　リージョンの関係性

表 6.2　リージョン A からの RTT

	平均値	最小値	最大値	標準偏差
リージョン A	0.234 ms	0.193 ms	0.271 ms	0.022
リージョン B	0.324 ms	0.287 ms	0.349 ms	0.015
リージョン C	0.321 ms	0.285 ms	0.347 ms	0.016
リージョン D	0.315 ms	0.277 ms	0.339 ms	0.016

6.3　保存ピア探索の計算コスト

データの保存，取得時に，Synapse では対象のデータの時間毎に保存ピアの探索を行う一方で，T-Ring は対象のセンサーの chunk，SP の値により，探索の回数が変化する．そこで，1000 個の連続したデータを保存，取得することを想定し，Synapse，T-Ring の両手法において，1 データの平均のピア間ホップ数を計測する．また T-Ring においては，各データの時間の間隔を 5，SP を 100，chunk を 10，30，50，70，90 の 5 つとする．図 6.2 はその結果である．

6.4　データ保存時の評価

データの保存時に Synapse では，データ毎に保存場所の計算を行う．その一方で，T-Ring では，ある一定の時間が経過するまでは，保存場所の変更が発生しない．そこで，ある保存ピアに

表 6.3　リージョン B からの RTT

	平均値	最小値	最大値	標準偏差
リージョン A	0.327 ms	0.288 ms	0.347 ms	0.015
リージョン B	0.238 ms	0.203 ms	0.270 ms	0.019
リージョン C	0.319 ms	0.274 ms	0.349 ms	0.018
リージョン D	0.317 ms	0.282 ms	0.347 ms	0.015

表 6.4　リージョン C からの RTT

	平均値	最小値	最大値	標準偏差
リージョン A	0.325 ms	0.277 ms	0.359 ms	0.016
リージョン B	0.312 ms	0.279 ms	0.342 ms	0.015
リージョン C	0.252 ms	0.192 ms	0.379 ms	0.022
リージョン D	0.319 ms	0.283 ms	0.343 ms	0.015

表 6.5　リージョン D からの RTT

	平均値	最小値	最大値	標準偏差
リージョン A	0.296 ms	0.240 ms	0.727 ms	0.047
リージョン B	0.295 ms	0.259 ms	0.374 ms	0.014
リージョン C	0.252 ms	0.192 ms	0.379 ms	0.022
リージョン D	0.291 ms	0.239 ms	0.335 ms	0.013

100 個のデータを保存するクエリを送り，それらの保存が完了するまでの時間を計測する．本評価では，保存されるデータの設定を chunk=30，SP=100，各データの時間の間隔を 5 として実験を行った．また，ネットワークの環境については，ネットワークに参加している保存ピアの数（ネットワークサイズ= NS）を 10，50，100 の 3 種類，各ピア間の RTT を 5，10，50，1000 の 4 種類とした．図 6.3，6.4，6.5，6.6 は RTT ごとにまとめた実験結果である．

6.5　データ取得時の評価

保存時と同様に，データの取得時に Synapse では，取得するセンサデータごとに保存場所の探索を行い，T-Ring ではある一定の時間が経過するまで，取得先の変更は発生しない．そこで，ある特定の領域に対して，500 単位時間のデータの取得を行うクエリを送り，データが全て返されるまでの時間を計測する．本評価では，データはすべて 5 単位時間ごとに保存されている環境を想定

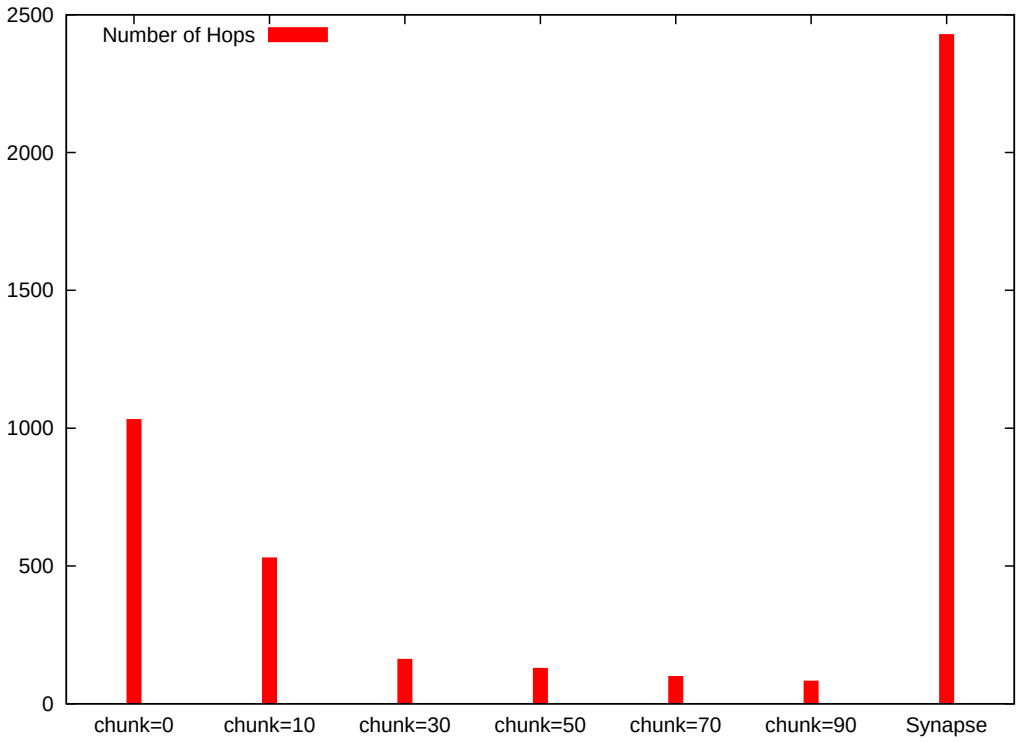


図 6.2　ホップ数

する．よって，500 単位時間分のデータは，100 個のデータを取得することとなる．クエリの地理的範囲は半径 10 の広さを持ったものとする．また，ネットワークに関しては，保存時と同様に，ネットワークに参加している保存ピアの数（ネットワークサイズ＝ NS）を 10，50，100 の 3 種類，各ピア間の RTT を 5，10，50，1000 の 4 種類とした．図 6.7，6.8，6.9，6.10 は RTT ごとにまとめた実験結果である．

6.6　考察

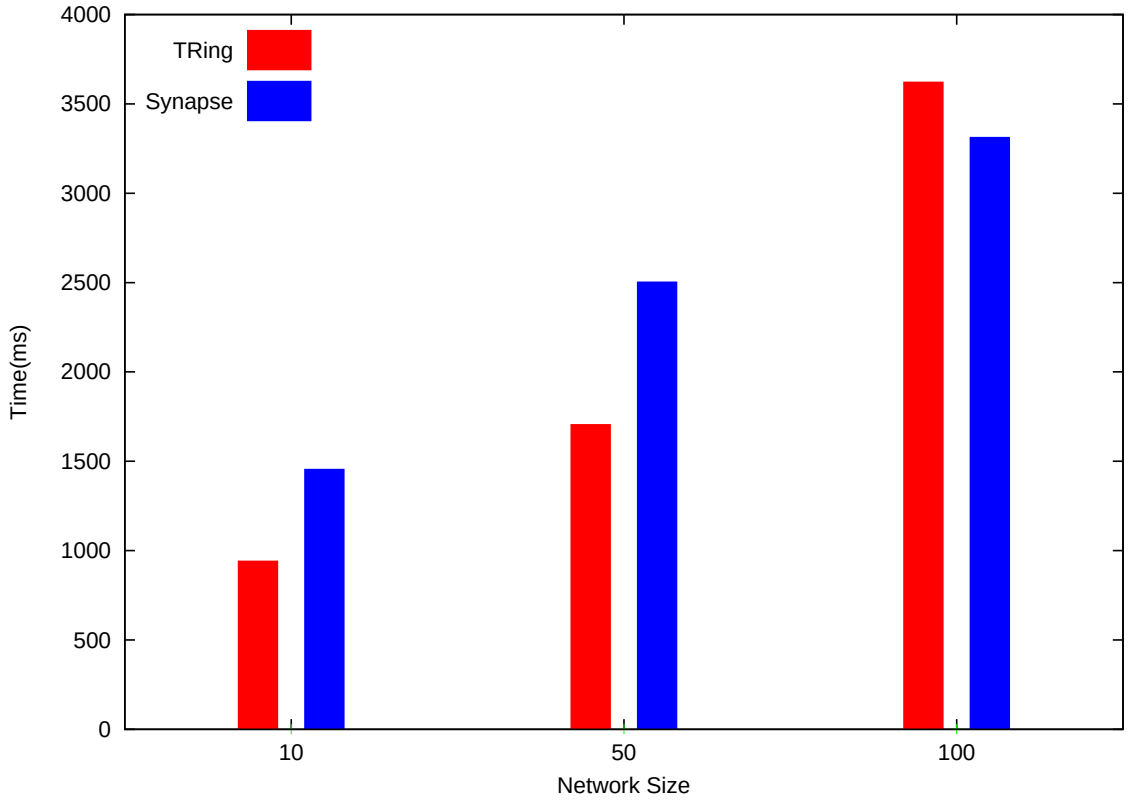
本節では，保存ピア探索の計算コスト，保存，取得それぞれの実験の結果に関しての考察を行う．

6.6.1　保存ピア探索の計算コストにおける考察

Synapse では，保存される全てのデータに対して保存ピアの計算が行われる．つまり，ネットワークサイズ＝ N，データの数＝ D，ホップ数＝ H とすると理論値では，

$$H = D \cdot \log 2N \tag{6.1}$$

となる．(6.1) における $\log 2N$ は，Chord の平均探索ホップ数である．それに対して T-Ring では，Synapse での所与の変数に加え，chunk の時間＝ C，SP の時間=S，保存されるデータの時間



	NS=10	NS=50	NS=100
T-Ring	944 ms	1708 ms	3625 ms
Synapse	1458 ms	2506 ms	3315 ms

図 6.3　保存：RTT=5

間隔=IT，保存されるデータの総時間=AT と，時間的な変数を加えて考えると，理論値は，

$$D = \frac{AT}{IT} \tag{6.2}$$

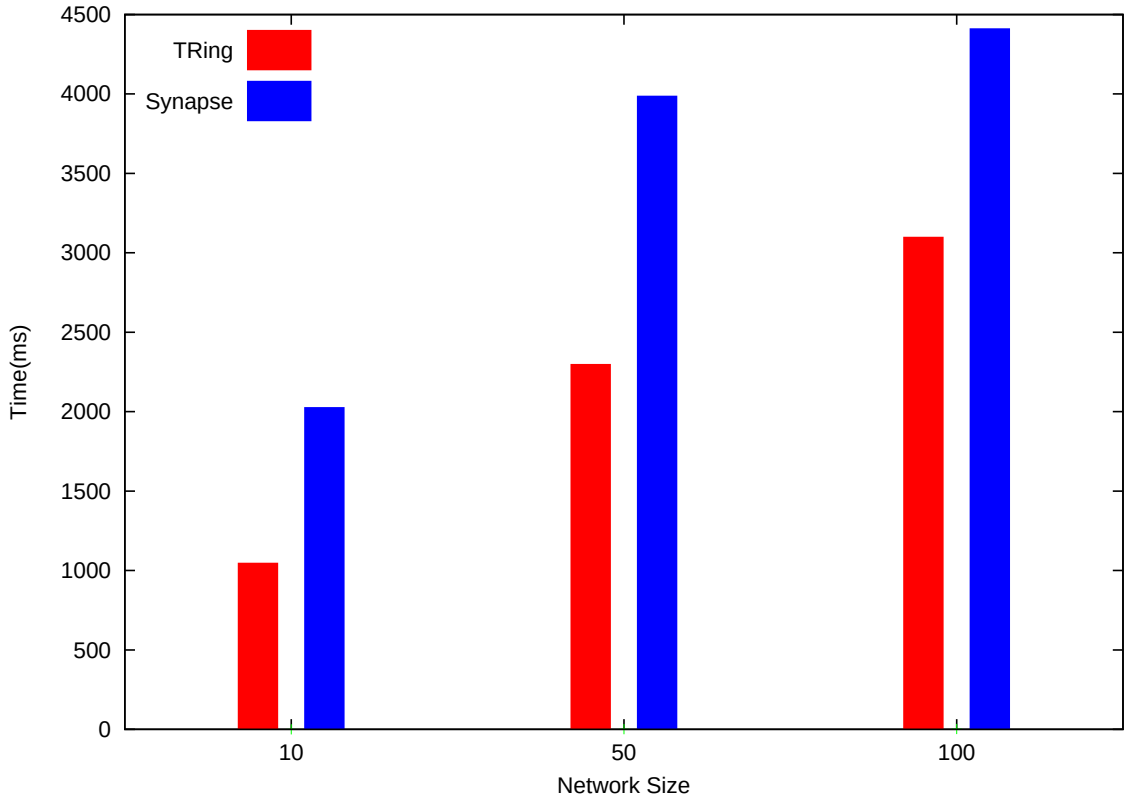
$$H = \frac{AT}{C} - \frac{AT}{S} + \frac{AT}{S} \cdot \log 2N \tag{6.3}$$

となり，(6.2)(6.3) から，AT を消去すると，

$$H = \frac{IT \cdot D\{S - C(1 - \log 2N)\}}{C \cdot S} \tag{6.4}$$

(6.4) となる．(6.1)(6.4) から，Synapse と T-Ring におけるホップ数（それぞれ SynapseH，T-RingH) の割合を計算すると，

$$\frac{SynapseH}{TRingH} = \frac{1}{IT} \cdot \frac{S \cdot C \cdot \log 2N}{S + C(\log 2N - 1)} \tag{6.5}$$



	NS=10	NS=50	NS=100
T-Ring	1049 ms	2300 ms	3101 ms
Synapse	2029 ms	3989 ms	4414 ms

図 6.4　保存：RTT=10

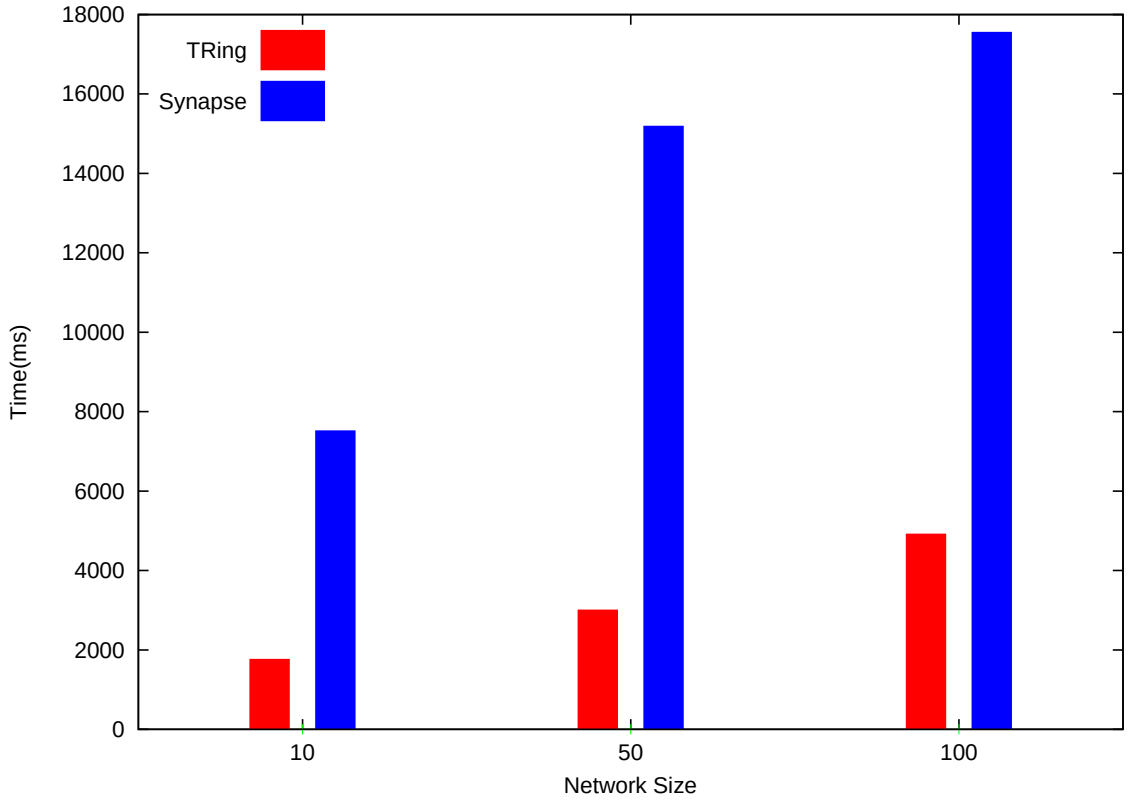
ここから，chunk と SP の関係性について考えると，

$$C \cdot S \cdot \log 2N - S + C(\log 2N - 1) = C \cdot \log 2N \cdot (S + 1) - (S + C) \tag{6.6}$$

と $C > S > 0$，$N > 0$ から，SP の値が大きくなる場合，chunk の値が SP の値に近づく場合，(6.5) の値が大きくなる．また，同様に IT の値が小さくなるほど，(6.5) の値が大きくなる．

ここから，実験結果を考察すると，実験結果は，理論値とほぼ相違ないことが分かる．よって，今回は chunk，SP，データの保存間隔を以上で述べた通りに行ったが，chunk=999，SP=1000，データの保存間隔=1 にした場合，Synapse のホップ数に限りなく近づく．

この保存ピア探索の計算コストは，データの保存，取得に大きく関係する．次節からこの関係性について考察する．



	NS=10	NS=50	NS=100
T-Ring	1049 ms	2300 ms	3101 ms
Synapse	2029 ms	3989 ms	4414 ms

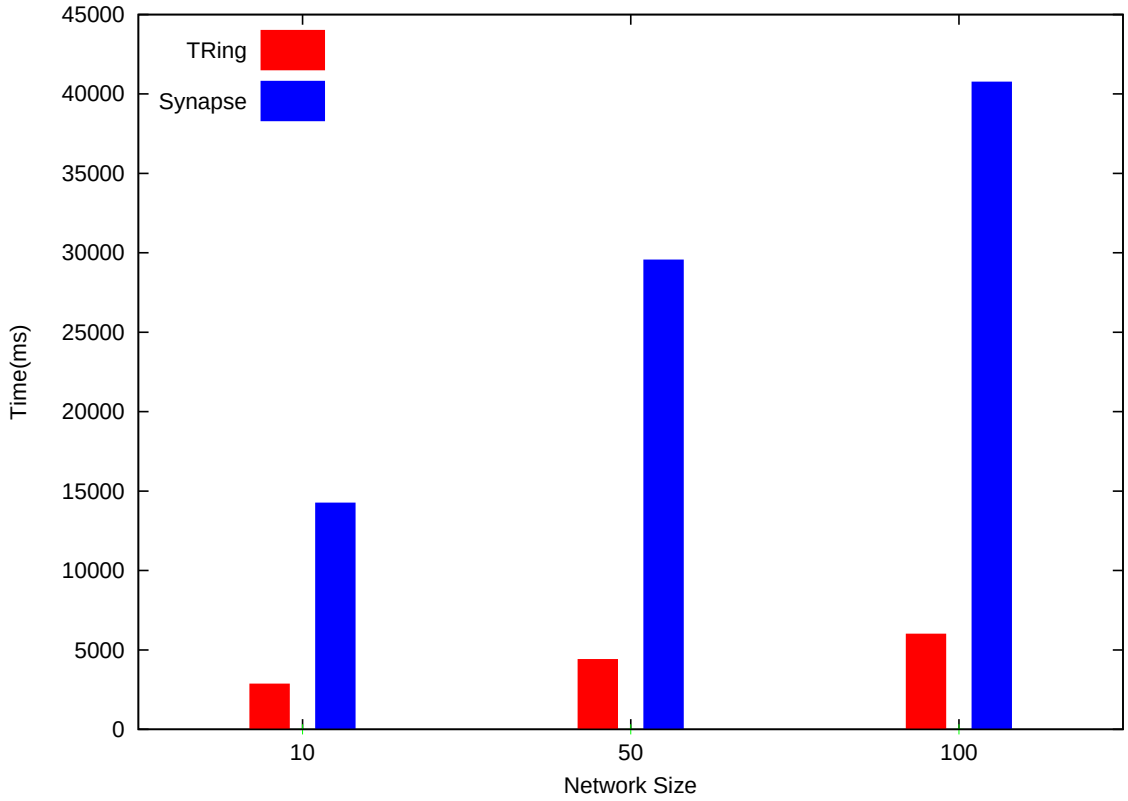
図 6.5　保存：RTT=50

6.6.2　データ保存における考察

前節において，保存ピア探索の計算コストについての考察を行ったが，Synapse と T-Ring のデータの保存にかかる時間の割合は，ネットワークレイテンシ（RTT）=RTT とすると，

$$\frac{SynapseH}{TRingH} \cdot RTT \cdot D \tag{6.7}$$

となる，Synapse と T-Ring を比較した際，RTT と D は同一の環境で実験をした場合，値の違いはないので，$\frac{SynapseH}{TRingH}$ において，取得時間は決定する．ここから，実験結果を考察すると，T-Ring における取得時間が理論値以上であることが伺える．これは，センサデータの保存を行う前段階のセンサ情報の取得による影響であると考えられる．本システムでは，センサデータがセンサノードから保存ピアに送られた際，そのセンサデータの時間とそのセンサノードの設置時間から，対象の



	NS=10	NS=50	NS=100
T-Ring	1049 ms	2300 ms	3101 ms
Synapse	2029 ms	3989 ms	4414 ms

図 6.6　保存：RTT=100

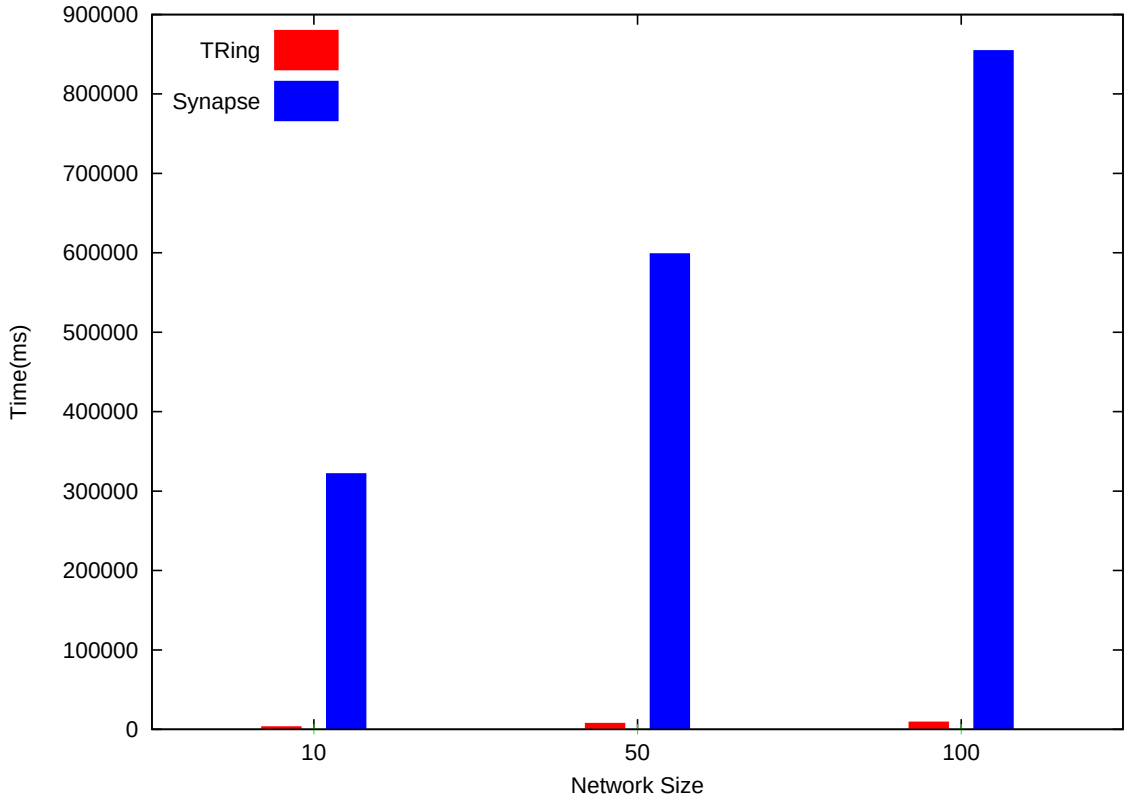
データの保存先が次の保存ピアに移行するのか，移行する際，異なるマスターノードに移行するのかを計算する．この計算に時間を要する分，T-Ring が理論値と異なっていると考えられる．

6.6.3　データの取得における考察

データの取得においても，保存時と同様に考え，クエリの検索範囲の半径=R とすると，Synapse と T-Ring のデータの保存にかかる時間の割合は，

$$\frac{SynapseH}{TRingH} \cdot RTT \cdot D \cdot \pi r^2 \tag{6.8}$$

となる．保存時と比較すると，対象範囲内の全ての座標を計算するため，πr^2 倍の差が発生することとなる．このことが，実験結果において，Synapse と T-Ring の時間の差が発生した論拠であると考えられる．

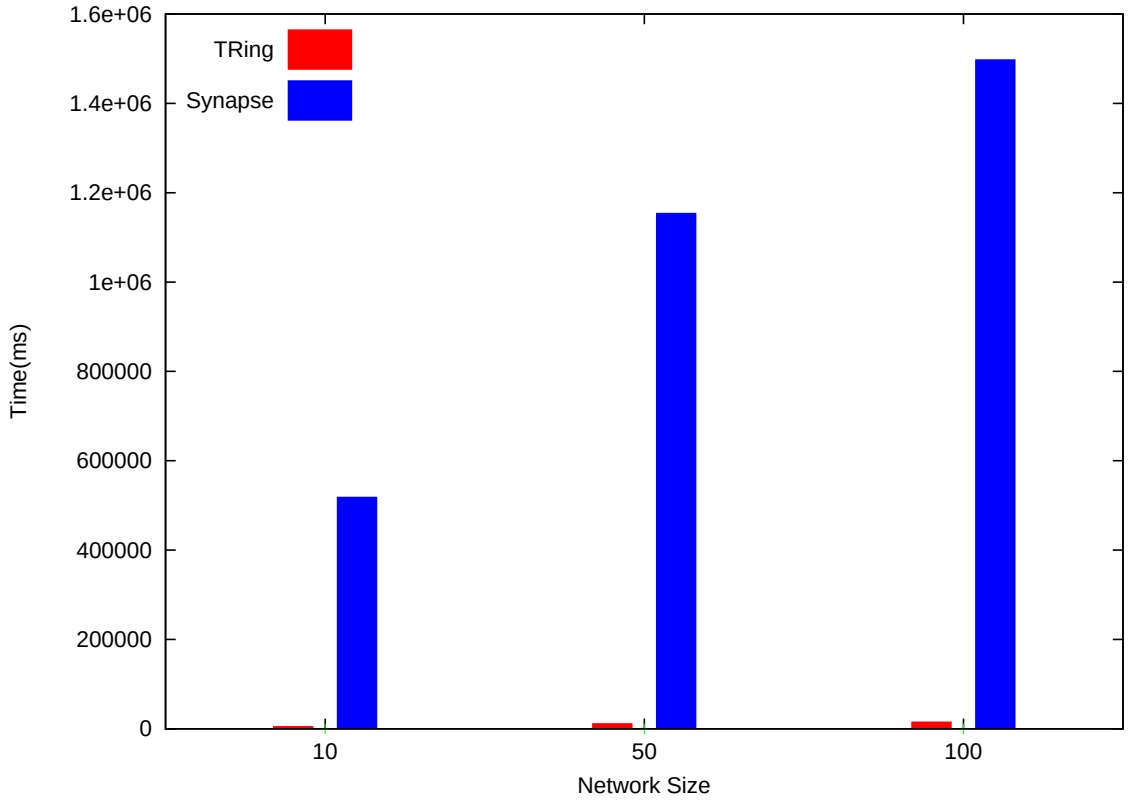


	NS=10	NS=50	NS=100
T-Ring	3884 ms	8098 ms	9823 ms
Synapse	322615 ms	599557 ms	855187 ms

図 6.7　取得：RTT=5

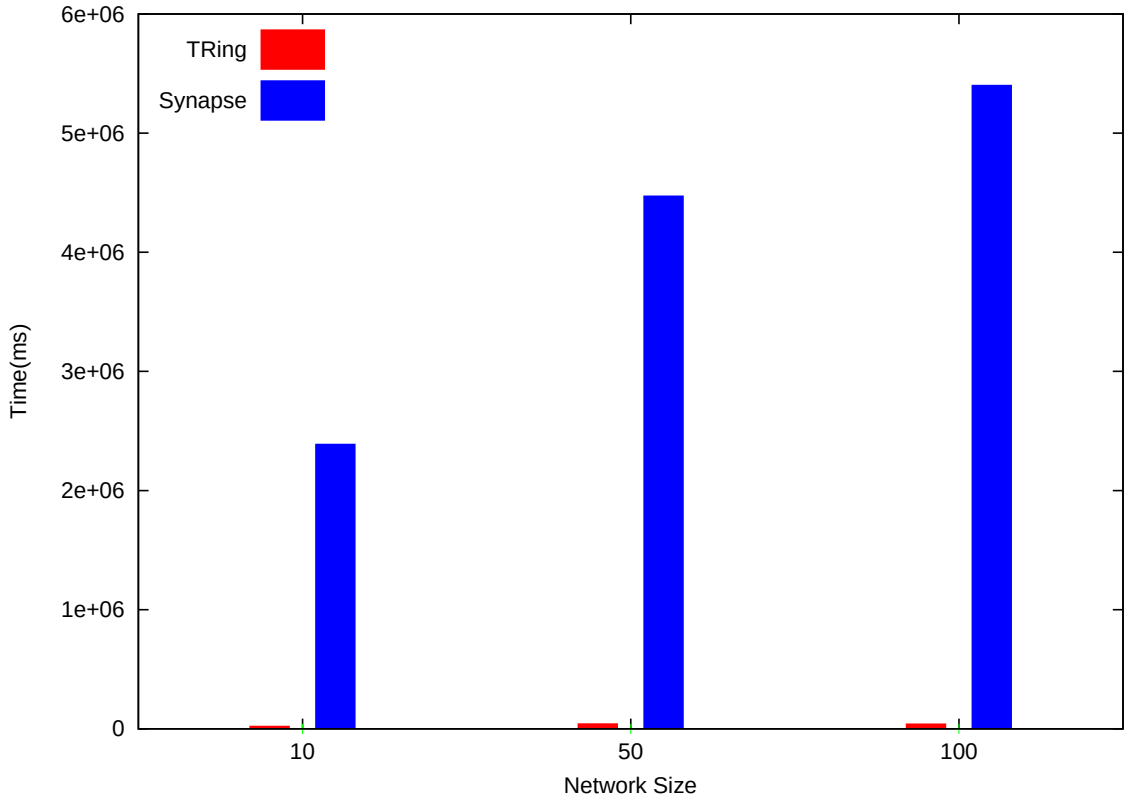
6.7　まとめ

本章では，まず，本研究に評価方針，実験環境を述べた．次に，実験環境内における実験への影響を調査するため，それぞれのリージョンから ICMP における，Message Request を送信し，リプライが返されるまでの時間（RTT）を計測した．その結果，RTT は 0.400ms 以内であり，十分に無視できることを述べた．次いで，保存ピア探索における計算コスト，データの保存に要する時間，データの取得に要する時間の観点から評価を行った．その結果から，考察を行い，理論値と実際の値に齟齬が存在することを明らかにし，その違いは，T-Ring において，保存や取得の対象となるデータの時間情報の取得を行う際に要する時間により生じていることに言及した．



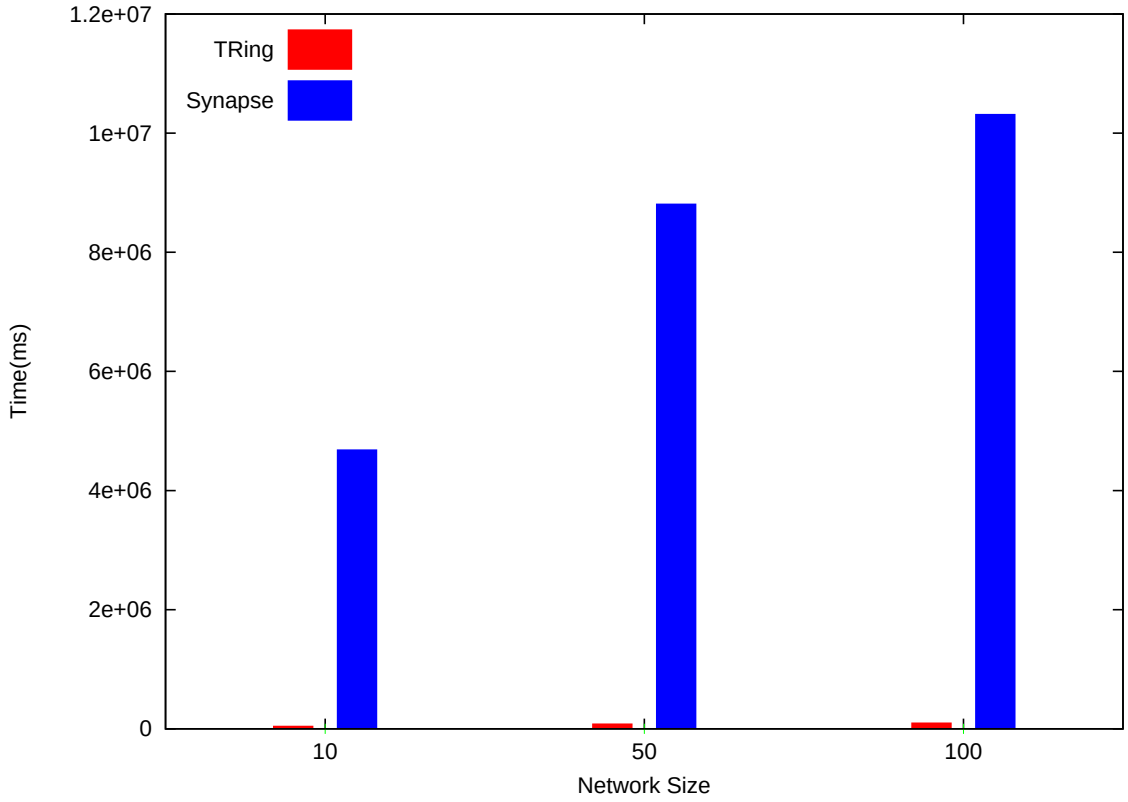
	NS=10	NS=50	NS=100
T-Ring	6247 ms	12683 ms	16249 ms
Synapse	519541 ms	1155219 ms	1498550 ms

図 6.8　取得：RTT=10



	NS=10	NS=50	NS=100
T-Ring	25730 ms	46740 ms	45068 ms
Synapse	2392797 ms	4477108 ms	5406327 ms

図 6.9　取得：RTT=50



	NS=10	NS=50	NS=100
T-Ring	51478 ms	90249 ms	104476 ms
Synapse	4691619 ms	8816386 ms	10322459 ms

図 6.10　取得：RTT=100

第 7 章

結論

本章では，T-Ring システムの結論を述べる．

7.1 今後の課題と展望

本研究では，センサデータの時間的特殊性に着目することにより，保存ピア探索における計算コスト，データ保存に要する時間，データ取得に要する時間について，既存の手法と比較し，それぞれ良い知見が得られた．しかし，データの取得に要する時間に関しては，依然として，課題が残る．既存手法との比較では，時間短縮に成功したが，半径 10 の領域から 100 個のデータの取得を行うに際し，時間数万 ms の時間を要している．想定するシナリオに挙げたように，ユーザがスマートフォンから情報を取得し，リアルタイムにアプリケーションに反映させることを斟酌すると，この所要時間では到底満足できない．よって，今後は，このデータの取得に着目した研究が必要となる．これに関する指針として，センサデータ管理のマルチレイヤ化がある．現行の T-Ring システムは，全てのデータを生データのまま保存している．しかし，現実に取得されるデータは，10:00，11:00 など切りの良い時間から取得される可能性が高いと考え得る．また，値の平均や最大，最小などが利用されるケースも存在する．このように，個々のセンサデータについての特徴は存在しないながらも，ある一定のデータが集約されることにより，特定の意味を有する 1 つのデータになることは十分に考えられる．これらの，データの集約による特徴ごとに，マルチレイヤで管理することにより，ユーザのクエリに対する対象データ数の削減に寄与すると考えられる．

また，本研究のメインフォーカスとして挙げてはいないが，近年，Cyber-Physical Systems と呼ばれる，実空間情報を情報空間に取り込むことにより，新たな価値を提供することを目指す研究分野が注目されている．Cyber-Physical Systems では，人間や物の行動や動きを逐次記録する手法が用いられることがある．この記録データは，センサデータと同様に時間に伴って増大する．T-Ring はこのようなデータを管理する手法としても用いることが可能である．

7.2 本論文のまとめ

本研究は，従来の多次元データ管理手法がセンサデータにおける時間のような，パラメータに特徴のあるデータを対象とした手法が存在しないことに着目した．次いで，センサデータの時間的特殊性に着目した T-Ring システムの提案を行った．このシステムの評価を行った結果，既存のシステムと比べ，データ保存，取得において，高速化を実現した．しかし，データの取得に関しては，リアルタイムアプリケーションに利用するに耐えうる所要時間ではないので，最終章において，今後の課題として取り組むべきであることに言及した．

謝辞

本論文の執筆にあたり，親身になって丁寧にご指導して頂きました，慶應義塾大学環境情報学部教授徳田英幸博士に深く感謝致します．また，また，貴重なご助言を頂きました應義塾大学環境情報学部准教授高汐一紀博士，慶応義塾大学環境情報学部専任講師中澤仁博士，慶応義塾大学環境情報学部米澤拓郎特任助教，慶応義塾大学政策・メディア研究科伊藤友隆氏に深く感謝致します．

慶應義塾大学徳田研究室の諸先輩方には折に触れ貴重なご助言を頂き，また多くの議論の時間を割いて頂きました．特に，政策・メディア研究科博士課程生天目直哉氏には，本研究に対し，多くの時間を割いて頂きました．ここに多大なる感謝と尊敬の意を表します．

また，数少ない同学年として，研究活動に切磋琢磨した，伊藤瑛氏，小鷲麻奈美氏，グエンミンザン氏，KMSF，Link 研究グループにおいて，研究活動だけではなく，公私に関わらず，親しく接していただいただいた井村和博氏，鈴木幸大氏，皆川昇子氏，高木慎介氏，宇佐美真之介氏，豊田智也氏，荻野メリッサ氏を始めとした諸先輩，後輩方，研究で疲弊している中，いつも強引に徹夜でのサッカーゲームに誘ってくれて，疲れを増幅させながらも精神衛生を整え続けてくれた二神直也氏，阿部寛氏に深く感謝し，謝辞と致します．

2014 年 12 月 3 日

寺山淳基

参考文献

[1] Volker Gaede and Oliver Günther. Multidimensional access methods. Vol. 30, pp. 170–231, New York, NY, USA, June 1998. ACM.

[2] Roger Weber, Hans-Jörg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24rd International Conference on Very Large Data Bases*, VLDB '98, pp. 194–205, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[3] Ibrahim Kamel and Christos Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, pp. 500–509, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[4] Sylvia Ratnasamy, Brad Karp, Li Yin, Fang Yu, Deborah Estrin, Ramesh Govindan, and Scott Shenker. Ght: a geographic hash table for data-centric storage. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, WSNA '02, pp. 78–87, New York, NY, USA, 2002. ACM.

[5] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pp. 149–160, New York, NY, USA, 2001. ACM.

[6] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '01, pp. 161–172, New York, NY, USA, 2001. ACM.

[7] James Aspnes and Gauri Shah. Skip graphs. In *Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '03, pp. 384–393, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[8] Chong Zhang, Weidong Xiao, Daquan Tang, and Jiuyang Tang. P2p-based multidimensional indexing methods: A survey. Vol. 84, pp. 2348–2362, New York, NY, USA, December 2011. Elsevier Science Inc.

[9] Akamai Technologies. `http://www.akamai.co.jp/`.

[10] Meeyoung Cha, Alan Mislove, and Krishna P. Gummadi. A measurement-driven analysis of information propagation in the flickr social network. In *Proceedings of the 18th international conference on World wide web*, WWW '09, pp. 721–730, New York, NY, USA, 2009. ACM.

[11] Yahoo! Inc. `http://www.flickr.com/`.

[12] Salvatore Scellato, Cecilia Mascolo, Mirco Musolesi, and Jon Crowcroft. Track globally, deliver locally: improving content delivery networks by tracking geographic social cascades. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pp. 457–466, New York, NY, USA, 2011. ACM.

[13] Twitter Inc. `https://twitter.com/`.

[14] Facebook. `http://www.facebook.com/`.

[15] Lu-An Tang, Xiao Yu, Sangkyum Kim, Jiawei Han, Wen-Chih Peng, Yizhou Sun, Hector Gonzalez, and Sebastian Seith. Multidimensional analysis of atypical events in cyber-physical data. In *Proceedings of the 2012 IEEE 28th International Conference on Data Engineering*, ICDE '12, pp. 1025–1036, Washington, DC, USA, 2012. IEEE Computer Society.

[16] Craig Labovitz, Scott Iekel-Johnson, Danny McPherson, Jon Oberheide, and Farnam Jahanian. Internet inter-domain traffic. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pp. 75–86, New York, NY, USA, 2010. ACM.

[17] Gregor Maier, Anja Feldmann, Vern Paxson, and Mark Allman. On dominant characteristics of residential broadband internet traffic. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pp. 90–102, New York, NY, USA, 2009. ACM.

[18] John S. Otto, Mario A. Sánchez, David R. Choffnes, Fabián E. Bustamante, and Georgos Siganos. On blind mice and the elephant: understanding the network impact of a large distributed system. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, pp. 110–121, New York, NY, USA, 2011. ACM.

[19] Junki Terayama, Jin Nakazawa, and Hideyuki Tokuda. Dht-based sensor data management for geographical range query. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pp. 623–624, New York, NY, USA, 2012. ACM.

[20] Jonathan K. Lawder and Peter J. H. King. Using space-filling curves for multi-dimensional indexing. In *Proceedings of the 17th British National Conferenc on Databases: Advances in Databases*, BNCOD 17, pp. 20–35, London, UK, UK, 2000. Springer-Verlag.

[21] Verena Kantere, Spiros Skiadopoulos, and Timos Sellis. Storing and indexing spatial data in p2p systems. Vol. 21, pp. 287–300, Piscataway, NJ, USA, February 2009. IEEE Educational Activities Department.