センサデータの高鮮度化・周期的発信・時系列処理を実現するデータベースシステムに関する研究

平成16年度

川島 英之

論文要旨

本論文は，センサデータを扱うアプリケーションを支援するデータベースシステムの実現方式および過負荷制御方式に関する研究をまとめるものである．

本論文の第 1 提案は，センサデータを高鮮度化・周期的発信・時系列処理するデータベースシステムの設計方式である．センサデータの使用目的が継続した実世界の認識であることに着目して，その目的を実現するデータベースシステムとして設計した KRAFT の構成を示す．まずセンサデータの高鮮度化を実現するために，リモートメモリを用いた Write Ahead-Logging(WAL) 方式および冪等なリカバリ方式を示す．リモートメモリの使用に際しては，ログ増加に伴うメモリ不足が問題となる．同問題を解決するための機構として，高速チェックポインタ機構と専用メモリ管理機構の実現方式を示す．次にセンサデータの周期的発信を実現するために FreeBSD の KSE に基づく pthread スケジューラを基盤として実時間スケジューラを設計する．そしてセンサデータの時系列処理を実現するために，センサデータに対する類似シーケンス検索演算をもつ抽象データ型により関係データモデルを拡張する．拡張されたデータモデルを満足するように，ストレージ管理機構・メモリ管理機構・クエリ処理機構を設計した機構の構成を示す．

評価実験を通して，KRAFT がセンサデータの高鮮度化・周期的発信・時系列処理を実現することを示す．まずセンサデータの高鮮度化に関する提案方式について，センサの発生周期が 10 ミリ秒である時に，監視周期が 1 秒のモニタへ 5 ミリ秒程度の鮮度を提供できた実験結果を示す．次にセンサデータの周期的発信に関する提案方式について，モニタの並行度が 1000 の場合に，モニタの開始時刻と予定時刻のずれがラウンドロビン方式よりも 1/279 にまで小さくなった実験結果を示す．さらに，センサデータの時系列処理に関する提案方式について，ユークリッド距離と DTW 距離に基づく類似シーケンス検索を実現し，SQL に基づく言語により検索できた実行例を示す．

本論文の第 2 提案は，データベースシステムの負荷を減らすために，センサデータの WAL 処理を軽量実行する方式である．モニタが監視していないセンサデータについて，WAL 処理におけるログ保存の確認処理を省くことにより，システム負荷の減少を実現する方式を示す．このアプローチは従来研究では扱われてこなかった．

専用実験システムにおいて，提案方式は TCP を用いたリモート WAL 方式に比べて最大 32%優れた鮮度を提供できた実験結果を示す．

以上より，本論文はセンサデータの高鮮度化・周期的発信・時系列処理を実現するデータベースシステムの実現方式を明らかにし，かつ，センサデータを扱うデータベースシステムのための新たな過負荷制御方式を示したと結論する．

Abstract

This dissertation proposes (1) design methodology of a database system for applications that deal with sensor data and (2) an overload resolution technique.

The first proposition of this dissertation is the design of a database system KRAFT that provides high freshness, periodic monitoring, and time-series processing of sensor data. I have designed KRAFT by considering that sensor data are used to recognize real-world continually. To realize high freshness of sensor data, I propose a Write Ahead-Logging(WAL) technique with a remote memory and an idempotent recovery technique. To solve memory shortage on a remote memory incurred by rapidly incoming log records, I have designed a fast check pointer mechanism and a memory management mechanism. To realize periodic monitoring of sensor data, I have designed a real-time scheduler based on FreeBSD KSE pthread scheduler. To realize time-series processing of sensor data, I have designed an expansion of the relational data model by incorporating an abstract data type that provides similar sequence retrieval operations.

I have evaluated how KRAFT provides high freshness, periodic monitoring and time-series processing of sensor data through experiments. As for high freshness of sensor data, I conducted an experiment with a monitor. Period of the monitor was 1 second and period of sensor data was 10 milli seconds. As a result of experiment, KRAFT could provide sensor data of which freshness is 5 milli seconds. As for periodic monitoring of sensor data, the KRAFT scheduler provides 279 times smaller gap between planned start time and real start time compared with the round robin scheduler. As for time-series processing of sensor data, KRAFT realizes similar sequence retrieval methods with the Euclidian distance and the DTW distance by using a SQL based language.

The second proposition of this dissertation is a light and imprecise WAL processing of sensor data to reduce load of a database system. By executing light and imprecise WAL processing with sensor data that are not read by periodic monitoring, this technique reduces heavy load. This approach has not been studied by any existing related work.

On a dedicated experiment system, the proposed method demonstrates 32 % better freshness of sensor data compared with remote memory WAL technique of which protocol is TCP.

From the results of experiments, this dissertation concludes that proposed studies have shown (1) design methodology of a database system for applications that deal with sensor data and (2) an overload resolution technique.

目 次

図 目 次

表 目 次

第1章

序論

1.1. データベースシステムの新たな課題：センサデータ

データベースシステムとは，計算機に実現される記録保持システムであり，その目的は情報の記録・共有・維持・検索・更新・統合である．このシステムの存在意義は，データの集中制御によるアプリケーション開発の効率化である [Date 84]．データベースシステムを使うことにより，様々なアプリケーションで必要なデータ管理という機能をアプリケーション毎に開発する必要がなくなる．このときアプリケーションの性能低下は避けられないが，データ管理機能の実装に必要なコストを考えればそれは取るに足らない問題であろう．2003 年度に 136 億ドル程もデータベースシステムが購入された事実はそれを裏付けているとも言える [IDC 03]．

今日データベースシステムは，我々の生活の様々な部分で使われている．例えば学校のデータベースが扱うデータには，教員の業績，学生の成績，卒業論文，修士論文，図書館の書籍，そして卒業生の個人情報などがある．これらのデータは，長期間に渡り幾度も閲覧されることを目的として，一昔前には紙媒体で記録保持されていたデータである．記録媒体が計算機に変わろうともデータの使われ方に変わりはない．

一方，瞬間的には使われるけれども，記録保持されることなく破棄されてきた種類のデータがある．そのようなデータは**センサデータ**と呼ばれている．センサデータは，コミュニケーションロボット [Kanda *et al.* 02]，自動車 [赤松 03]，環境モニタリング [Mainwaring *et al.* 02]，モーションキャプチャ [大崎 他 99, 本田 03]，バイラテラルシステム [Katsura *et al.* 04] 等の**センサ応用システム**で使われている．

センサデータを記録保持・解析処理できれば，人間社会をより豊かにする技術を次のように開発できるであろう．コミュニケーションロボットでは，長期間にわたり人間の嗜好に適応するペットロボットの開発ができるであろう．自動車では，故障の早期診断や危険状態の予測を行うシステムを開発できよう．環境モニタリングでは，国立公園における鳥類の生態系を詳細に解析できるようになるであろう．モーションキャプチャでは，動作によるコンピュータへの指令入力インタフェース

を構築できるであろう．バイラテラルシステムでは，熟練外科医師の手術スキルを測定することで初心者の手術技術を教育支援するシステムを開発できるであろう．

すなわち，センサデータを記録保持できるようになれば，様々な有益な技術の開発を支援できるようになる．

1.2.　本研究の目的

本研究の目標は，センサ応用システムを支援するデータベースシステムを開発することである．前述のセンサ応用システムがセンサデータを使う理由を考えてみると，継続して外界状況を認識するシステムの実現が重要になると考えられる．

それならばデータベースシステムには次の機能が求められる．まず，センサ応用システムが継続して外界状況を知るために，センサデータを周期的に継続して提供することが求められる．また，センサ応用システムは，現在の外界状況を認識する必要があるために，提供するセンサデータが新鮮であることも求められる．さらに，センサ応用システムが信号にすぎないセンサデータから状況を認識することを支援するために，センサデータの時系列表現と，様々なシステムで認識に使われる手法である，類似シーケンス検索手段の提供も求められる．これらの要求に加えて，センサ応用システムがインターネットを通じて多数のユーザによって使われるようになれば，突発的に発生する予測不能な過負荷状態にも対応する必要がある．以上の課題は次のように簡潔に表現できる．本研究はこれら 4 つの課題を解決することを目的とする．

- センサデータの高鮮度化
- センサデータの周期的発信
- センサデータの時系列処理
- 過負荷制御

1.3.　従来研究とそれらの問題点

本研究の目的を実現するにあたって必要になる機能はセンサデータの高鮮度化，センサデータの周期的発信，センサデータの時系列処理，そして過負荷制御である．

本研究の目的を包括的に達成する既存技術は存在しないが，4 要素の各面に注目した技術は存在する．ここではそれらを紹介すると共に，それらを本研究の目的に適用することができるか検討する．

1. センサデータの高鮮度化に関する従来研究

センサデータを高鮮度化する際のボトルネックは永続化処理に要するコストであるため，永続化処理を高速化できればセンサデータを高鮮度化できる．高速永続化処理に関する主要な従来研究には，P*TIME や ClustRa 等の主記憶関係データベース管理システム (MMRDBMS) がある．P*TIME はログファイルを N 個のディスクに分けて永続化処理を高速化する．ClustRa は 2 相コミットプロトコルを用いてリモートメモリにログを置くことで永続化処理を高速化する．

MMRDBMS のデータ永続化方式は優れたものであり，センサ応用システムに適用できると考えられる．

仮に MMRDBMS を用いて本研究の目的を達成しようとすれば，MMRDBMS は設計を根本的に修正する必要がある．周期的発信導入のためにスレッドスケジューラ，時系列データ型導入のためにストレージ構成，というシステムコアを変更する必要がある．

2. センサデータの周期的発信に関する従来研究

   周期的発信に関する主要な従来研究は 2 種類ある．第 1 に継続的に SQL クエリを実行することを目的に開発されているデータストリーム管理システム (DSMS) に関する研究がある．第 2 にトランザクションを実時間で処理することを目的に開発されている実時間データベースシステム (RTDBMS) に関する研究がある．

   しかしながら，センサデータの周期的発信に関して，いずれの研究にも問題があると考えられる．DSMS の問題は，周期的にクエリを実行可能であるものの，RTDBMS のように実時間処理を考慮していないために周期性の精度が低いと考えられる点である．RTDBMS の問題は，トランザクションの実時間性が考慮されているものの，想定されるトランザクションはデータベースシステムの外部から到着するものであり，データベースシステム内部での継続的クエリ実行が考慮されていない点である．

   仮に DSMS や RTDBMS を使って本研究の目的を達成しようとすれば，DSMS でも RTDBMS でも設計を根本的に修正する必要がある．なぜなら時系列データ型導入のためにストレージ構成，高速データ永続化導入のためにトランザクション管理機構，というシステムコアを変更する必要があるからである．

3. センサデータの時系列処理に関する従来研究

   センサデータの時系列処理に関する従来研究には，関係データベースシステムに抽象データ型として時系列データ型を追加する時系列関係データベース管理システム (TRDBMS) と，時系列認識処理を行う様々な類似シーケンス処理システム (SSPS) がある．

しかしながら，センサデータの時系列処理に関して TRDBMS には問題があると考えられる．その問題とは統計演算を提供するものの認識処理手法を提供しない点である．SSPS はいずれも優れた認識手法を提供するため，時系列処理の観点においては問題ないと考えられる．

仮にこれらの研究を用いて本研究の目的を達成するならば，TRDBMS も SSPS も設計指針に周期的発信と高速永続化処理がないために設計を根本的に修正する必要がある．

TRDBMS に関しては，周期的発信導入のためにスレッドスケジューラ修正，高速データ永続化導入のためにトランザクション管理機構修正，そして類似シーケンス検索手法を導入する必要がある．SSPS に関しては，周期的発信と高速データ永続化をもつデータベースシステムとして設計しなおす必要があると共に，様々な SSPS で使われている類似シーケンス検索手法を導入する必要がある．

4. 過負荷制御に関する従来研究

過負荷制御に関する従来研究には大きく分けてアドミッション制御 [Datta *et al.* 97, Kang *et al.* 04] とインプリサイストランザクション処理 [Vrbsky *et al.* 93] がある．アドミッション制御は，一部のトランザクションの受け付けを拒否することで新規トランザクションによる負荷増大を防ぐ．インプリサイストランザクション処理は，トランザクション処理を不完全に終えることでトランザクション処理過程の負荷を減らす．

過負荷制御に関する従来研究の問題点は，更新トランザクションのインプリサイス処理が研究されてこなかった点である．アドミッション制御にもインプリサイストランザクション処理にも様々な従来研究があり，このなかでインプリサイストランザクション処理は，検索処理については着目されてきたものの，更新・追加トランザクションに関するインプリサイス処理については考えられてこなかった．しかし，センサデータの挿入を行うセンサトランザクションは，データベースへのデータ挿入前に必ず永続化デバイスへの書き込み処理を必要とする Write-Ahead Logging(WAL) により小さくない負荷をデータベースシステムへ与える．そして多数のセンサから頻繁に発生するセンサデータを永続化する場合，WAL の負荷は急激に増大する．それゆえ WAL 処理過程に踏み込むことによりセンサ応用システムを支援するデータベースシステムの過負荷制御に関する性能を向上させることが期待される．

1.4.　本研究の方針

前節ではセンサデータの高鮮度化，センサデータの周期的発信センサデータの時系列処理，そして過負荷制御に関する従来研究を俯瞰した．それらの従来研究には，ひとつの問題については適用可能な手法もあったが，複数の問題に適用可能な手法はなかった．その理由を端的に述べるならば，従来研究がセンサ応用システムを明確に意識してこなかったためだと考えられる．そこで本論文ではセンサ応用システムを念頭において本研究の目的を達成するために 2 つの方針をとる．

1.4.1.　第 1 方針

本研究の第 1 方針は，センサデータの高鮮度化・周期的発信・時系列処理の 3 要素を単一のデータベースシステム上で実現することである．従来研究の問題点は，それぞれを個別に考えてきたことである．これら 3 つの要素の設計はいずれもシステムコアに大きく影響するため，各要素を個別に切り分けてきた従来研究は本研究の目的達成には適用困難である．高鮮度化はトランザクション処理機構に影響し，周期的発信はスレッドスケジューラに影響し，時系列処理はメモリ/ ディスクでのデータ配置方式に影響する．ただし過負荷制御は切り分けが可能であり，それを第 1.4.2 節で述べる．

センサ応用システムを支援するデータベースシステムを実現するために解くべき課題は，3 要素を統合したデータベースシステムの実現方式である．そして 3 要素それぞれに関する本研究の指針を以下に示す．

1. センサデータの高鮮度化

   センサデータの鮮度を高めるためには，センサデータがデータベースシステムに到着してからそれをクライアントへ提供可能にするまでに要する時間を短縮する必要がある．そして，このときに行なわれる処理はデータの永続化である．そこで永続化処理である WAL をネットワーク越しのリモートメモリに実行することで永続化処理を高速化する．

   通常はディスク上の単一ファイルに全トランザクションのログを書く．この性質のために多くのトランザクションが同時にアクセスした場合，ファイルロックを持たないトランザクションは待たざるを得ない．しかし，ネットワークを使えば全トランザクションを並行処理できるから，永続化処理を高速化でき，データ鮮度を高めることができる．

2. センサデータの周期的発信

   周期的にデータベースを監視するために，一定周期毎の規定時刻にエグゼキュータを実行する機構を，データベースレベルとスケジューラレベルで実

現する．規定時刻にエグゼキュータを実行する理由は，一定周期毎のデータベースのスナップショットを提供したいからである．規定時刻きっかりにエグゼキュータを呼び出せれば，正確に一定周期毎のデータスナップショットを提供できる．これは応答時間がデッドライン以内ということではない．データを監視する時刻が X+A, X+2A, X+3A... となる，ということである．

ラウンドロビンスケジューリングを用いては，スケジューリングポリシとの不整合から，一定周期でエグゼキュータを呼び出すことが困難である．そこでスケジューラを修正することにより一定周期毎にエグゼキュータを呼び出す機構を実現する．

3. センサデータの時系列処理

センサ応用システムがセンサデータを効率的に処理できるよう，本研究では関係データモデルの属性としてセンサデータを管理する時系列型を導入し，時系列型に対しては状況認識に有効な一手法である類似シーケンス検索を提供する．

関係モデルでセンサデータを扱うとすると，日付や名前といった非センサデータとセンサデータをマッピングをするために，結合演算を含む膨大な計算コストがかかる．それを避けるために，非センサデータは関係データとして扱い，センサデータは時系列データとして扱い，そして両者を円滑に統合するデータモデルを導入する．

1.4.2.　第 2 方針

本研究の第 2 方針は，WAL に注目した過負荷制御方式を提案し，専用実験システムにおいてその性能を評価することである．本研究では今までは注目されなかった WAL の処理過程における過負荷制御を新たに提案し，その有効性を専用実験システムを通して示す．ただし本方式は手法の検証にとどめ，開発は DBMS に組み込まない．

私がこの方式をデータベースシステムに組み込まない理由は，本研究で提案する過負荷制御方式の実現には少なくない負担を要するにも関わらず，組み込むことによる効果がそれほど大きくないことである．詳細は第 2.4.2 節で述べる．

1.5.　論文の構成

本論文の構成は次の通りである．第 2 章では本研究の背景として，ターゲットであるセンサ応用システムを述べ，論文で用いる言葉を定義し，問題を定式化し，問題に対する本研究のアプローチを述べる．第 3 章では関連研究として，センサデータの高鮮度化，センサデータの周期的発信，センサデータの時系列処理，そして過負

荷制御に関する従来研究を述べる．第 4 章では問題解決の第 1 アプローチとして，センサデータの高鮮度化・周期的発信・時系列処理を実現するデータベースシステム KRAFT について述べる．第 5 章では問題解決の第 2 アプローチとして，センサデータのインプリサイス永続化処理による過負荷処理技法を提案する．第 6 章ではそれまでの論文の内容を振り返り議論を行う．最後に第 7 章では結論と展望を述べ，本論文をまとめる．

第2章

背景

本章では本論文のターゲットとするセンサ応用システムについて述べ，本論文で用いる言葉を定義し，本論文で挑む問題を定式化したあと，最後に本研究のアプローチを示す.

2.1. センサ応用システム

センサを用いた処理を行うセンサ応用システムには様々なものがあるが，本節ではそのなかでモーションキャプチャ，地震データ監視，コミュニケーションロボット，そして環境モニタリングについて述べる.

2.1.1. コミュニケーションにおける相互適応メカニズム

本田 [本田 03] は，人間の身体動作とロボットの行動との対応付けを行うことにより人間とロボット間の意図伝達におけるプロトコルを動的に生成する相互適応メカニズムを実現している.

この研究では光学式モーションキャプチャシステムを用いて人間の動作を時々刻々と取得し，それを HURMA と名付けられた独自の時系列データベースに格納する．そして現在得られた動作データと類似する動作をデータベース中に蓄えられている過去の動作から LCSS(Longest Common SubSequences) 距離に基づいて検索し，それをロボットの動作と対応付けることで，研究目的であるプロトコルの動的生成を達成している.

このセンサ応用システムでは，データを蓄えるだけではなく，実験中の人間の動作を監視したいために周期的監視機構が必要になる．そして監視中に得られたデータには当然ながら鮮度が高いことが求められる．最後に，データは時系列表現されて類似検索という動作認識手段が使用される.

それゆえ，本システムを支援するには (1) センサデータの高鮮度化，(2) センサデータの周期的発信，(3) センサデータの時系列処理，を実現するデータベースシステムが必要となる.

2.1.2.　地震データ監視

東京大学地震研究所では日本全国から発生するセンサデータを観測し，それを WEB で公開している [東大地震研 05]．これを図 2.1 に示す．観測地点は 1194 箇所であり，データ量は一日に 20 ギガバイト程度である．地震データ監視システムでは，将

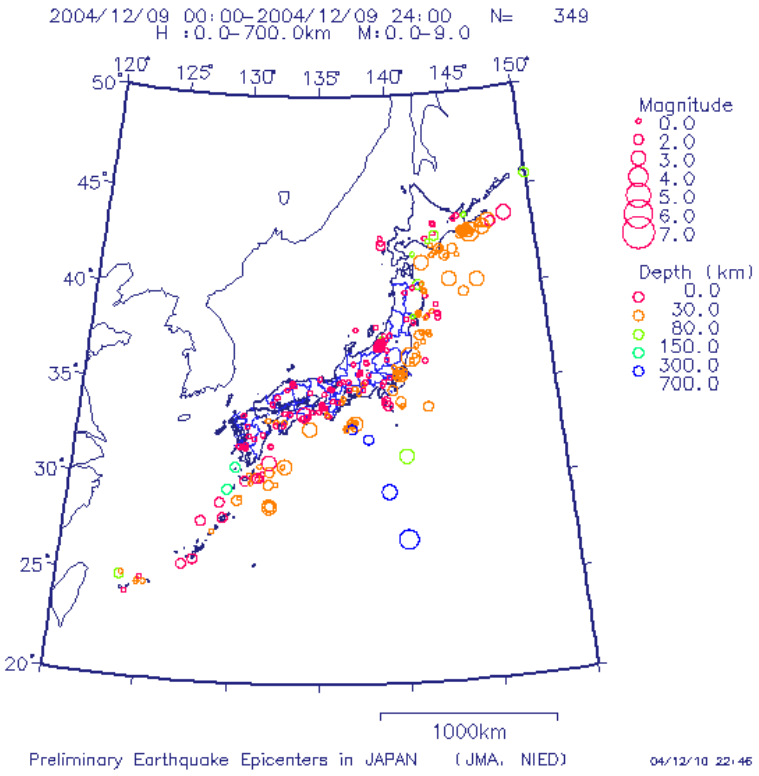


図 2.1: 日本付近の地震活動

来における解析をするために DBMS に到着したデータを永続化するばかりでなく，時々刻々と変化する地震データを監視することを望むため，データベースシステムはセンサデータの高鮮度化と周期的発信を提供する必要がある．さらに地域名などの非センサデータと地震データをマッピングするために，図 2.2 に示すような時系列データ表現とその時系列データに対する処理手法を必要とする．

図 2.2 に示すようなセンサデータの時系列表現が必要である理由は，センサデータを関係モデルで表現すると，図 2.3 に示すようにセンサ値ひとつひとつに対して非センサデータを対応させる必要があるからである．この場合，センサ値ひとつ毎に非センサデータを対応させる必要があるため，情報が冗長に保持される必要がある．図 2.3 では Relational Data もしくは Same data as above と記述されたタプルが冗長情報に該当する．両者を異なるテーブルに分けて管理すれば格納における冗長

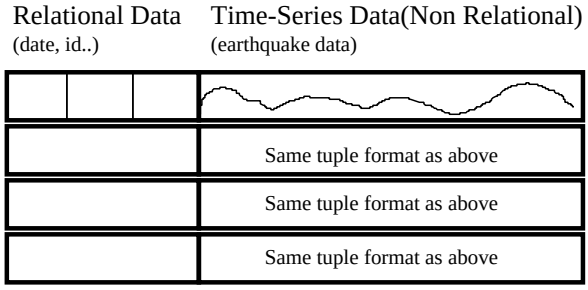


図 2.2: 地震データ管理例

性を解消できるが，両者をまとめて閲覧する場合には両方のテーブルを結合しなければならないから，結合演算という大きなコストの処理を実行する必要が生じてしまう．それゆえ関係モデルを用いるだけではセンサデータを効率的に管理できないと考えられる．

ところで時系列データに対する処理手法としては，一般的な統計演算に加えて類似地震波の検索処理のために類似時系列データを検索する機能が要求される．それゆえデータベースシステムは類似時系列データ検索機能を含むセンサデータの時系列処理手法を提供する必要がある．それから，将来的にインターネットを通じてシステムを多くの人に閲覧させることを考えると，閲覧人数が増えて負荷が急激に高まった場合の過負荷制御方式も考えられる必要がある．

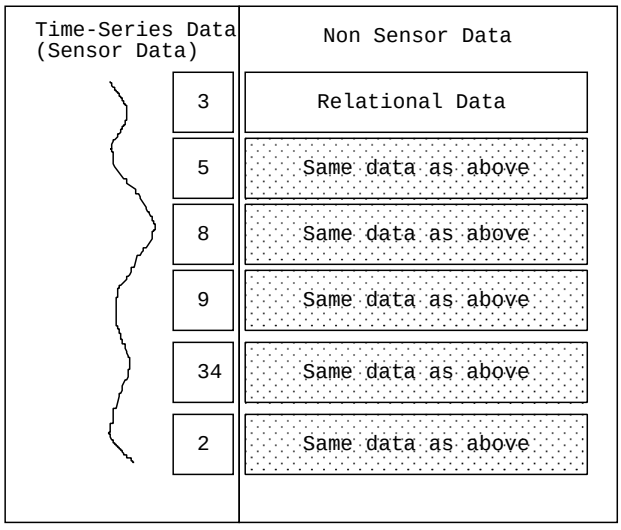


図 2.3: 関係データモデルによるセンサデータ表現

それゆえ地震データ監視システムを支援するには，(1) センサデータの高鮮度化，(2) センサデータの周期的発信，(3) センサデータの時系列処理，を実現するデー

タベースシステムが必要となると同時に，(4) 過負荷制御方式の検討が必要になる．

2.1.3. コミュニケーションロボット

ロボットによる人間支援システムの研究をするために，ATR 知能ロボティクス研究所は Robovie-II[Kanda *et al.* 02] と呼ばれるコミュニケーションロボットを開発している．以下，簡略のためにこれを Robovie と表記する．Robovie は図 2.4 に示されるヒューマノイド型ロボットである．Robovie の仕様を表 2.1 に示す．

図 2.4: コミュニケーションロボット: Robovie

コミュニケーションロボットのセンサから得られるデータは，ロボットとインタラクションした人間の状態を表す．これは将来において解析を行うことによりロボットの行動プログラムを改善するために保存しておく必要がある．それだけでなく，人間・ロボットインタラクションの実験を行う認知心理学者は，実験中においてもデータを観察し続けることを求める．それゆえデータベースシステムにはセンサデータを高鮮度化する機構と周期的なデータ発信機構が必要となる．地震データ監視システムと同様に，実験データを効率的に管理するためには日付などの非センサデータとセンサデータのマッピングおよび解析処理が必要である．なぜなら両者を関係モデルのみで表現すると，センサデータの表現は地震データ同様に図 2.3 に示すようなテーブル構成にならざるを得ないと同時に解析処理が提供されないからである．それゆえデータベースシステムにはセンサデータを時系列データとして処理する時系列処理が必要になる．それから将来的に実験システムをインターネットを通じて多くの解析者・認知科学者に閲覧させることを考えると，閲覧人数が増えて負荷が急激に高まった場合の過負荷制御方式も考えられる必要がある．

表 2.1: Robovie の仕様

寸法	高さ 114cm 幅 52cm 奥行き 50cm
重量	39kg
最大移動速度	1.6m/s
腕の最大運動速度	200 °/s
バッテリー	DC12V 21Ah
距離センサ	超音波センサ 24 個
タッチセンサ	感圧導電性ゴムタイプ 16 個、スイッチタイプ 2 個
バンパセンサ	スイッチタイプ 10 個
ステレオカメラ	SONY EVI-G20 2 個
全方位カメラ	1 個
ステレオマイク	SENNHEISER MKE104 1 個
両腕モータ	ハーモニックドライブ DC モータ 8 個
頭部モータ	ハーモニックドライブ DC モータ 3 個
移動機構	2 輪独立駆動車輪およびキャスター
内蔵 CPU	Pentium III 850MHz
メインメモリ	128MB
OS	Linux2.2.16-rtl2.2
通信装置	無線 LAN

それゆえコミュニケーションロボットを支援するには，(1) センサデータの高鮮度化，(2) センサデータの周期的発信，(3) センサデータの時系列処理，を実現するデータベースシステムが必要となると同時に，(4) 過負荷制御方式の検討が必要になる．

2.1.4.　環境モニタリング

さまざまなセンサを配置してデータを収集することにより，環境状況を監視するセンサネットワークが増えつつある．たとえば MOTE[XBOW 04] と名付けられたセンサノードがある．CrossBow 社から販売されている MOTE には幾つかの種類があるが，現在購入可能なものでは，500 円玉サイズのものが最小である．MOTE 上では専用 OS である TinyOS が用意されており，TinyOS は他の MOTE とアドホックネットワークを形成する機能をもつ．さらに SQL インタフェースを備えた軽量フィルタである TinyDB もある．TinyDB はデータベースシステムのクライアントとして動作するソフトウェアであり，それ自身がデータを永続的に蓄えることは考えていない．センサネットワークの例としては，国立公園内で鳥の存在検出をするシステム [Mainwaring *et al.* 02] や兵士の状態認識を行うシステム [Tatbul *et al.* 04] 等がある．

さて，センサネットワークを構築する目的は鳥の存在検出や兵士の状態認識などの特定状況の検出であるため，センサ応用システムとしては継続してセンサデータを監視する必要がある．それゆえデータベースシステムにはセンサデータの周期的発信が必要になる．センサ応用システムは，得られたデータはその瞬間に発生したものだと考えるだろうから，このときに提供されるデータには高い鮮度が必要とされる．それゆえデータベースシステムにはセンサデータの高鮮度化が必要になる．さらに得られたセンサデータからミクロ・マクロの両観点から環境状況の解析をすることも考えられる．これを支援するにはセンサデータの時系列処理が必要になる．そして長期解析のためにはデータを保存して置く必要があるから，データが永続化される必要とされる．それから将来において構築した環境モニタリングシステムをインターネットを通じて多くの人に閲覧させることを考えると，閲覧人数が増えて負荷が急激に高まった場合の過負荷制御方式も考えられる必要があろう．

それゆえ，(1) センサデータの高鮮度化，(2) センサデータの周期的発信，(3) センサデータの時系列処理，を実現するデータベースシステムが必要となると同時に，(4) 過負荷制御方式の検討が必要になる．

2.2.　用語定義

本節では本論文で用いる言葉を定義する．

定義 2.1 (センサデータ)

本論文では，あるセンサデータを s と表記し，s を次のように定義する．

$$s = <at, v>$$

ここで at はセンサデータがデータベースシステムに到着した時刻 (Arrival Time) を表し，v はセンサデータの値 (Value) を表す．$s = <at, v>$ から at と v を得る演算を，それぞれ $\mathsf{at}(s)$，$\mathsf{v}(s)$ と表記する．

また，連続的にセンサデータを監視する処理であるモニタが読める s を s_{read} とし，データベースシステムがモニタに読ませない s を s_{unread} と表記する．s_{unread} は過負荷制御のために第 5 章で用いられる．

定義 2.2 (センサデータストリーム)

本論文では，あるセンサデータストリームを S と表記し，S を s の集合と定義する．さらに，s_{unread} を含む S を S_{unread}，s_{read} のみを含む S を S_{read} と表記する．S_{unread} および s_{unread} は第 5 章で用いられる．

定義 2.3 (リモートメモリ)

本論文では，リモートホスト上のメモリを**リモートメモリ**と定義する．

定義 2.4 (永続性)

本論文では，永続性を p と表記し，p を「データベースシステムが障害によって停止しても，再起動における復旧処理によりシステムから消失しない性質」と定義する．さらに p の中で，確実に保証される p を p_{strong} と表記し，確実には保証されない p を p_{weak} と表記する．p_{strong} と p_{weak} は過負荷制御に関わる概念であるために第 5 章で扱うが両概念の必要性を簡潔に述べておく．第 5 章では，リモートメモリを利用したセンサデータの永続化処理を述べる．このとき，リモートホストからの ack を要求するセンサデータには p_{strong} が与えられ，それ以外のセンサデータには p_{weak} が与えられる．

そして,「s に対して p を与える処理」を $\mathsf{p}(s)$ と定義する．$\mathsf{p}(s)$ の結果，s に p_{strong} が与えられることを「$\mathsf{p}(s) \rightarrow p_{strong}$」と表記する．同様に，$\mathsf{p}(s)$ の結果，s に p_{weak} が与えられることを「$\mathsf{p}(s) \rightarrow p_{weak}$」と表記する．

当然リモートメモリも障害によってデータを失うが，第 5 章では複数台のリモートメモリにログレコードを書き込むことでログレコードが失われることを防ぐ手法を考える．

定義 2.5 (鮮度)

s をすでにコミットされたセンサデータとする．本論文は s の古度と鮮度をそれぞれ次のように定義する．ただし，$\mathsf{at}(s)$ は s がデータベースシステムに到着した時刻であり，$\mathsf{rt}(s)$ は s がトランザクションにより読まれた時刻である．このとき，鮮度 (freshness) と古度 (staleness) を次のように定義する．

$$\mathsf{staleness}(s) = \mathsf{rt}(s) - \mathsf{at}(s) \tag{2.1}$$

$$\mathsf{freshness}(s) = \frac{1}{\mathsf{staleness}(s)} \tag{2.2}$$

2.3.　問題の定式化

本節では問題の定式化をおこなう．解決すべき問題は，(1) センサデータの高鮮度化，(2) センサデータの周期的発信，(3) センサデータの時系列処理，そして (4) 過負荷制御である．(1)〜(3) はデータベースシステムの深部に関わる問題であるため，3 つを同時に解決するデータベースシステムを実現することにより，同時解決を試みる．一方 (4) はデータベースシステムの深部に関わる問題ではないために，専用システムを開発しての評価を行う．アプローチの詳細については第 2.4 節で述べる．

2.3.1.　センサデータの高鮮度化

定義より，staleness(s) が小さいほど freshness(s) は高まる．そこで本論文ではセンサデータの高鮮度化 ($Q_{\text{高鮮度化}}$) を次のように定式化する．

$$Q_{\text{高鮮度化}} = \text{staleness}(s_{read}) \text{ の極小化} \tag{2.3}$$

2.3.2.　センサデータの周期的発信

周期 P でクエリを実行し，その結果をクライアントに発信することを，センサデータの周期的発信と定義する．

周期的発信での最初のクエリ実行時刻を T とし，周期的発信について，予定されるクエリ実行時刻 (Planned Query Time) を $PQTE_i = T + P, \ldots, T + NP$($N$ は整数) とする．一方，周期的発信について現実にクエリが実行される直前の時刻 (Real Query Execution Time) を $RQTE_i$ とする．

このときセンサデータの周期的発信に関する課題 ($Q_{\text{周期的発信}}$) を次のように定式化する．

$$Q_{\text{周期的発信}} = (RQTE_i - PQTE_i) \text{ の極小化} \tag{2.4}$$

2.3.3.　センサデータの時系列処理

センサデータを時系列データモデル TDM により管理し，それ以外のデータを関係データモデル RDM により管理することで，RDM に存在しない演算を TDM に対して効率的に実行できる．なぜならば TDM をもたない従来の関係 DBMS で RDM に存在しない演算を実行するには，次の 3 つの処理が必要であるために計算コストが必要になるからである．(処理 1) センサデータを図 2.3 に示した非効率形式で保持すること．(処理 2) データベースシステムの API を通じてそれらをアプリケーションへ転送すること．(処理 3) 求める演算を実行すること．センサデータの時系列処理に関する演算は広く存在するが，本研究では認識に幅広く使われる処理である類似シーケンス検索を実現する演算を用意する．

さて，これよりセンサデータの時系列処理に関する問題の定式化を行う．長さ N であるふたつのシーケンスを S_A と S_B とし，S_A の構成要素である i 番目のセンサデータを S_{A_i} と表記する．

このとき S_A と S_B のユークリッド距離を次のように定義する．

$$Dist_{euclid}(S_A, S_B) = \frac{\sqrt{\sum_{k=1}^{N}(S_{A_k} - S_{B_k})^2}}{N} \tag{2.5}$$

次に文献 [Keogh *et al.* 99] に従ってダイナミックタイムワーピング (DTW) 距離を定義する．まず，要素が距離 $d(S_{A_i}, S_{B_j}) = (S_{A_i} - S_{B_j})^2$ である $N \times N$ のマトリックスを作成する．ただし $1 \leq i \leq N$, $1 \leq j \leq N$ である．ワーピングパス W はマトリックス $(1, 1)$ から始まり，最もコストの低いルートを選びつつ，(N, N) で終わる．選ばれたマトリクス点の値を w_k として，DTW 距離を次のように定義する．ただし w_k と w_{k+1} は縦横斜めのいずれかの方向について隣接しており，k が 1 つ増えたとき，i と j の少なくともいずれか片方は増加する．

$$Dist_{dtw}(S_A, S_B) = \min\left\{\frac{\sqrt{\Sigma_{k=1}^{K} w_k}}{K}\right\} \tag{2.6}$$

以上より，本論文ではセンサデータの時系列処理 ($Q_{\text{時系列処理}}$) を次のように定式化する．

$$Q_{\text{時系列処理}} = TDM \text{ に対する } Dist_{euclid}\text{・}Dist_{dtw}\text{の提供} \tag{2.7}$$

2.3.4.　過負荷制御

本論文では過負荷制御を次のように定式化する．

$$Q_{\text{過負荷制御}} = \mathsf{p}(s_{read}) \rightarrow p_{strong}\text{かつ } \mathsf{staleness}(s_{read}) \text{ の非増加} \tag{2.8}$$

2.4.　本論文のアプローチ

本論文では前述した 4 つの問題を解決するために 2 つのアプローチを採用する．

2.4.1.　2 つのアプローチ

本論文は上記のセンサ応用システムがデータベースシステムに求める機能である，$Q_{\text{高鮮度化}}$，$Q_{\text{周期的発信}}$，$Q_{\text{時系列処理}}$，$Q_{\text{過負荷制御}}$ について以下の 2 つのアプローチを採る．

- 通常状態へのアプローチ

  本論文は問題解決への第 1 アプローチとして，$Q_{\text{高鮮度化}}$，$Q_{\text{周期的発信}}$，$Q_{\text{時系列処理}}$ を個別に解決する技法を提案すると同時に，それらの技法を組み込んだデー

タベースシステムを設計し，その上で提案技法を評価する．これら 3 つの課題はデータベースシステムにとって常時要求される機能であるため，通常状態へのアプローチと記述する．

- 異常状態へのアプローチ

  本論文は問題解決への第 2 アプローチとして，$Q_{過負荷制御}$ を解決する技法を提案し，専用実験システム上でその技法を評価する．この課題はデータベースシステムにとって特殊な場合においてのみ要求される機能であるため，異常状態へのアプローチと記述する．

2.4.2. $Q_{過負荷制御}$ を分ける理由

上記のように本論文は 4 つの問題を解決する技法を全てデータベースシステムに組み込んで解決するのではなく，$Q_{過負荷制御}$ だけを分けて解決する．この理由は本研究で提案する過負荷制御方式の実現には少なくない負担を要するものの効果が極めて大きいわけではないためである．これを以下で詳述する．

過負荷制御方式は大まかに分類するとアドミッション制御とインプリサイストランザクション処理の 2 手法に分けられる．これらの実現コストと過負荷制御への効果を考えてみる．

アドミッション制御は一部のトランザクションについて受け付けを拒否し，拒否したトランザクションを破棄する方式である．これは破棄したトランザクションについては全く処理を行わないため，新規トランザクションによるシステム負荷の増大がなくなる．そのためアドミッション制御は過負荷制御に大きな効果があると考えられる．アドミッション制御の実現にはトランザクション開始部の修正が必要だが，この部分はデータベースシステムの表層に位置するために実現容易であると考えられる．

インプリサイストランザクション処理はトランザクション処理過程を不完全に実行する方式である．トランザクションの種類にはデータベースの内容を問い合わせる検索トランザクションとデータベースの内容を変更する更新・挿入トランザクションの 2 つがあるが，後者に関する研究は私が知る限り存在しないため，ここでは検索トランザクションに関するインプリサイス処理について考える．

検索トランザクションに関するインプリサイス処理は，ある検索トランザクションについてその全処理過程の一部のみしか実行しないことにより過負荷制御に効果を発揮する．しかしながらこの方式の実現には次のような負荷が生じる．(負荷 1) 検索実行方式および検索結果転送方式をインタラクティブにせざるをえないことによりプロセス間通信コストが増加すること．(負荷 2) 新規検索トランザクションの部分的実行による負荷増大が避けられないこと．それゆえインプリサイストランザクション方式は，トランザクションを破棄するアドミッション制御ほど過負荷制御に対する効果はないと考えられる．インプリサイストランザクション処理の実現に

はトランザクション処理過程およびトランザクション処理結果転送過程を修正する必要があるため，データベースシステムの中層を修正せざるを得ず，アドミッション制御よりも実現が困難である．

すなわち，アドミッション制御は効果が大きく実現容易である一方，インプリサイストランザクション処理は効果がそれほど大きくなく実現困難であると考えられる．本研究では後者のインプリサイストランザクション処理に着目した過負荷制御を提案する．そこで少ないコストで提案手法の有効性を検証するために，$Q_{過負荷制御}$ に関するアプローチについては専用実験システムでの実現および評価を行う．

以上の理由より，本論文では $Q_{高鮮度化}$，$Q_{周期的発信}$，$Q_{時系列処理}$ を同時に解決するアプローチと，$Q_{過負荷制御}$ を単独で解決するアプローチを採用する．

2.5.　本章のまとめ

本章では，まず研究をおこなう背景として近年増加の一途を辿るセンサ応用システムを紹介した．それらは相互適応メカニズムシステム，地震データ監視システム，コミュニケーションロボット，そして環境モニタリングだった．次に本論文で用いる言葉の定義を行った．それから本論文が取り組む 4 つの問題を，$Q_{高鮮度化}$，$Q_{周期的発信}$，$Q_{時系列処理}$，$Q_{過負荷制御}$ と定式化した．最後に本論文の問題解決アプローチを述べた．アプローチは 2 つあり，第 1 アプローチは $Q_{高鮮度化}$・$Q_{周期的発信}$・$Q_{時系列処理}$ を同時実現するデータベースシステムの提案だった．第 2 アプローチは $Q_{過負荷制御}$ を解決する手法を提案し専用実験システムでその評価をおこなうことだった．

第3章

関連研究

本章では，本研究の関連研究を述べる．本研究が取り組む問題は4つあり，$Q_{\text{高鮮度化}}$，$Q_{\text{周期的発信}}$，$Q_{\text{時系列処理}}$，$Q_{\text{過負荷制御}}$，と前章で定式化した．これらを全てもしくは複数を同時に課題とした研究はないが，一つについて課題として研究は多数存在する．本章ではそれらの従来研究に関して述べる．

3.1. データの高鮮度化

データの鮮度が最も高まる場合は，入ってきたばかりのセンサデータをモニタが読めた時である．これを実現するには，(1) センサデータがデータベースシステムに到着したら即座にモニタへ読めるよう準備する手法と，(2) センサデータを挿入するトランザクションの実行順序をスケジューリングする手法，の2つの手法がある．さて，手法 (1) においてモニタにセンサデータを読めるようにさせるには，センサデータが永続化される必要がある．さもなければ電源故障などの突発的システム停止後に，モニタが読んだデータがデータベースに存在しなくなってしまうかもしれない．そして手法 (2) においてトランザクション処理順序をスケジューリングするとは，モニタが求めるセンサデータを優先して処理するスケジューリングを意味する．

換言すればデータの高鮮度化を実現するには，(1) 永続化処理の高速化，(2) 鮮度を考慮したトランザクションスケジューリング，の2つの手法があると言える．このうちデータの高鮮度化への影響は (2) よりも (1) の方が大きい．なぜならば (1) は各トランザクション自身の処理コストに関連するからである．すなわち，いくらトランザクションをスケジューリングしようとも，トランザクション自体の処理が遅くては，スケジューリングによってデータを高鮮度化することはできない．そこで本節では，(1) の永続化処理に関する従来研究を述べる．

3.1.1.　永続化処理

データベースシステムに到着したデータはすべからく永続化される必要があるため，この処理時間が短いほど鮮度を高めることができる．本節ではこの永続化処理について説明する．

データベースシステムは障害による停止後にリスタートされた時，データベースの一貫性が破られないように復旧処理が行われる．復旧処理には大きく分けて 2 つの方法がある [Mohan 99]．それらは WAL とシャドーページングである．WAL は多くのシステムで使われているが，シャドーページングは SystemR のような一部のシステムでしか採用されていない．本論文では WAL に注目し，シャドーページングについては述べないが，その詳細は文献 [Gray *et al.* 01a] を参照されたい．特にシャドーページングの欠点は文献 [Gray *et al.* 01a] の 13.5.1.4 節にまとめられている．さて，WAL はデータベース領域にアクセスする前にログを WAL ファイルへ書く方式である．データベースシステムの復旧時には，WAL ファイルに記述されたログを使ってデータベースを一貫性のある状態まで戻す．通常この WAL ファイルはひとつしか存在せず，全てのトランザクションは同一 WAL ファイルへ書き込みを行う．そしてリカバリ時には WAL ファイルから各トランザクションがコミットされているか調べ，コミットされているトランザクションについては REDO と呼ばれる操作の再実行を行う．そしてコミットされていないトランザクションについては UNDO と呼ばれる操作の取消を行う．

ログレコードの形式には差分ログと状態ログの 2 種類がある．データベースシステムが管理するページについて，差分ログは前ページと新ページの排他的論理和をとる方式であり，状態ログは前ページと新ページをそのまま保存する方式である [Härder *et al.* 83]．ログレコードの形式は性能に大きく関与する．第 3.1.2 節では差分ログを用いた高速永続化について述べる．

3.1.2.　並列ディスクを用いた高速永続化

並列ディスクを用いた高速永続化を実現するシステムに P*TIME があり，これは Seoul National University で開発されている主記憶データベースシステムである [Lee *et al.* 01, Cha *et al.* 04]．

P*TIME のアーキテクチャを図 3.1 に示す．同図において，データは全てメモリ内の Primary DB 領域に納められ，ディスクには保存されない．従って，データ量が多い程，多くの Primary DB 領域が必要になり，それを納めるに足るほど多くのメモリが必要になる．同図における Backup DB はチェックポインタ ChckpntMgr により作成される Primary DB のバックアップである．Backup DB は ChckpntMgr によるチェックポインティング処理の度に作成され，そのとき Backup DB α は Backup DB $\alpha + 1$ と名前が変更される．さて，リカバリ時には Log と Backup DB 1 を用いて Primary DB が再生される．文献 [Lee *et al.* 01] では Backup DB 2 までが必須だ

とされているが，この理由は，Backup DB1 の名前を Backup DB 2 に変更してから Backup DB1 を作成している間にシステム故障が発生してしまった場合，その Backup DB 2 と Log を用いてリカバリを行うからであろう．

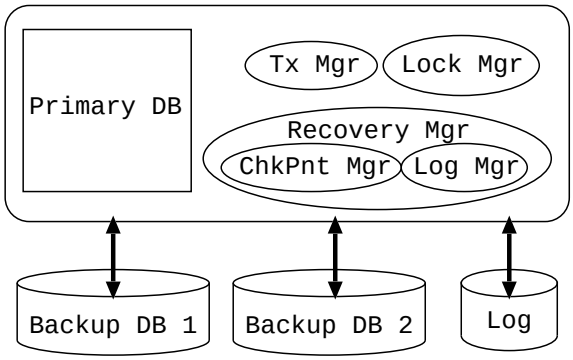


図 3.1: P*TIME のアーキテクチャ

P*TIME では WAL を高速化するために Differential Logging という手法を用いている [Lee *et al.* 01]．Differential Logging の特徴は 2 つある．第 1 特徴は差分ログ方式を用いることである．これによりログの量を物理ロギングを用いる場合の半分程度まで減らせる．なぜなら更新操作のログとして，ARIES 方式 [Mohan 99] の物理ロギングは before イメージと after イメージの両方を必要とするが，差分ロギングはそれらのビットレベルの XOR 差分しか必要としないからである．第 2 特徴は複数サーバを用いることで WAL ファイルを複数個用いることである．これにより WAL 実行時の負荷を分散し，WAL の実行を高速化する．通常は WAL ファイルを分割できないが，Differential Logging ではページ単位の差分ログの利用により分割を成功させている．

Differential Logging の利点は並列なリカバリ実行にある．並列なリカバリ実行を図 3.2 に示す．並列なリカバリ実行には表 3.1 に示すような 4 種類がある．表 3.1 において | はパイプライン実行，|| は並列実行，そして → は逐次実行を表す．このような並列実行ができる理由は，ページ毎にログ領域を設けることで各ログ領域への Backup Play を並列化すると同時に，差分ログの使用により Log Play を Backup Play と同時に実行できるからである．

P*TIME がセンサデータ処理に不向きな点は，全てのデータがメモリに常駐することが求められる点である．更新が頻発するだけのデータベースならばこの方式は適用可能だが，センサデータは更新ではなく追加が主たる操作であるために，時間と共にデータベースのデータ量は増加する．それゆえ P*TIME アーキテクチャはセンサデータに適用できない．文献 [Cha *et al.* 04] では，P*TIME が株式市場のデータ管理に使用された例が述べられているが，これは現時点のデータを参照できるだけであり，過去の履歴は全く扱えない．それゆえ P*TIME はアーカイブを解

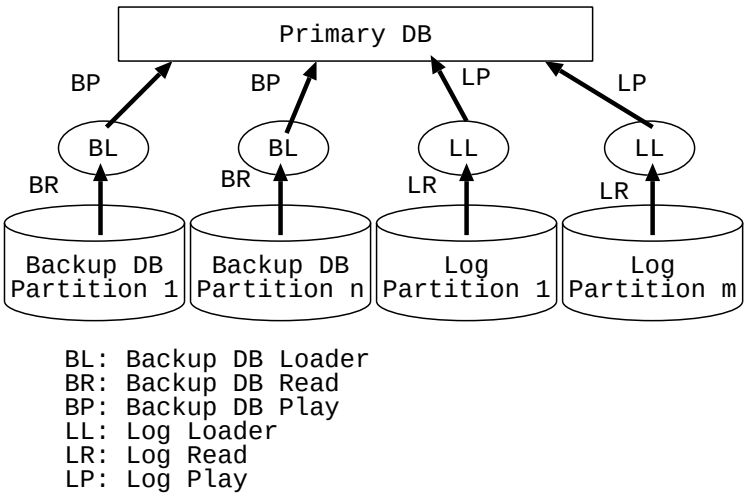


図 3.2: P*TIME の並列なリカバリ実行

表 3.1: 4 種の並列なリカバリ実行

レベル	並列性	リカバリ時間
L1	$\{BR\} \rightarrow \{BP\} \rightarrow \{LR\} \rightarrow \{LP\}$	$T_{BR} + T_{BP} + T_{LR} + T_{LP}$
L2	$\{BR\|BP\} \rightarrow \{LR\|LP\}$	$\max(\ T_{BR}, T_{BP}\ ) + \max(T_{LR}, T_{LP})$
L3	$(\{BR\|BP\} \| \{LR\}) \rightarrow \{LP\}$	$\max(\ T_{BR}, T_{BP}, T_{LR}) + T_{LP}$
L4	$\{BR\|BP\} \| \{LR\|LP\}$	$\max(\ T_{BR}, T_{BP}, T_{LR}, T_{LP})$

析して傾向分析をするといった要求には応えられない.

3.1.3. リモートメモリを永続的デバイスとする高速永続化技法

リモートメモリを永続的デバイスとする高速永続化技法として主記憶 WAL がある. この技法は通信・移動体ネットワーク用データを管理するために開発された ClustRa と呼ばれる主記憶データベースシステムにおいて開発された [Hvasshovd *et al.* 95]. ClustRa は無共有結合方式[1]の並列 DBMS でもあり，複数のノードをサーバとして立てることができる．ClustRa が用いる主記憶 WAL とは，ローカルディスクの代わりに 2 台の隣接ノードのメモリに 2 相コミットを用いてログを書く方式である. ローカルディスクを使わない理由は，ログをディスクに書くよりも隣接ノードのメモリにログを書く方が高速であるために WAL の高速化が見込まれるからである. 評価実験において 2 つの更新トランザクションをサーバ毎に実行した場合の応答時間が，サーバ数が 2 の場合は 14.6 ミリ秒であり，サーバ数が 8 の場合は 21.5 ミリ秒となったことが同文献には示されている.

ClustRa の永続化方式である主記憶 WAL はディスクを用いる WAL に比べて高性能を実現できると考えられる．ClustRa は永続化方式については優れるものの，これをセンサ応用システムに適用することは難しいと考えられる．なぜなら ClustRa はスイッチ上で使われることを目的として開発されているため，状況認識に必要な時系列処理や監視に必要な周期的発信について全く考慮されていないからである.

3.1.4. 高速永続化に関する従来研究のまとめ

高速永続化に関する従来研究として P*TIME の Differential Logging と ClustRa の主記憶 WAL があったが，このうち主記憶 WAL は Differential Logging よりも高い鮮度を提供すると考えられる．なぜなら主記憶 WAL はリモートホストのメモリを使うために全トランザクションについて WAL を同時実行できる可能性がある一方, Differential Logging はディスクを用いるためにディスク台数だけしか同時実行できないため，主記憶 WAL の永続化処理速度は Differential Logging の永続化処理速度を上回る可能性があると考えられるからである.

ところで P*TIME と ClustRa をセンサ応用システムに適用するために問題となる共通事項は 2 つあった．第 1 の問題は全てのデータが主記憶に格納できなければ動作しない前提条件だった．この前提条件は量システムが増え続けるセンサデータを扱えないことを表す．第 2 の問題は時系列処理と周期的発信が考えられていないことだった．それゆえ両システムはメモリが無限にあろうとも $Q_{\text{時系列処理}}$ と $Q_{\text{周期的発信}}$ を解決しない.

[1]無共有結合方式とはディスクもメモリも共有しない並列化方式である.

3.2.　周期的発信

周期的発信に関連する研究としては，データストリーム管理システム (DSMS) に関する研究と実時間データベースシステム (RTDBMS) に関する研究がある．

DSMS は，従来の DBMS は提供しないコンティニュアルクエリを効率的に実行させることを目的に設計されたデータ処理システムである．コンティニュアルクエリとは，一度だけ実行される通常のクエリは異なり，データベースシステム内部に一定期間留まってクエリ結果を生成し続けるクエリである．しかしながら DSMS はコンティニュアルクエリを周期的に実行させることを考えていないため周期的発信の精度は低いと考えられる．

RTDBMS は，従来の RDBMS に実時間性というパラメータを追加して，実時間トランザクションを実現しようとするシステムである．最初はトランザクション応答時間の実時間性保証のみが研究課題とされたが [Abbott *et al.* 88]，現在はそれに加えて RTDBMS が提供するデータの鮮度を保証することも研究課題になっている [Kang *et al.* 04]．

本節ではこれら DSMS と RTDBMS について述べる．

3.2.1.　データストリーム管理システム (DSMS)

本小節では 3 つの DSMS を取り上げる．いずれの DSMS もコンセプトは同じであるが，それぞれ特徴があり設計指針が異なる．

3.2.1.1.　STREAM

STREAM[Babcock *et al.* 02] は Stanford University で開発されている DSMS である．STREAM では 2 種類のデータを仮定している．ひとつは従来の DBMS であつかう永続的データであり，これらは関係データとして扱われ，かつトランザクションにより処理される．もうひとつは現在の状態を監視するために導入されたストリームデータである．ストリームデータはトランザクションにより処理されず，永続化されない．そしてそれはコンティニュアルクエリによる監視が行われる．

STREAM はストリームデータを扱うために SQL を拡張して CQL(Continuous Query Language) という宣言的言語を開発している．例えば CQL において過去 2 分間のデータを得るためには次のような記述をおこなう．

```
Select * From S [Range 2 Minutes]
```

ストリームデータには 3 種類の演算子が用意されており，それらは (1) ストリームデータから関係データへ変換する演算子，(2) 関係データから関係データへの演算子，そして (3) 関係データからストリームデータへの演算子である．(1) が必要な理由は，関係演算は有限長のタプルにしか適用できないが STREAM ではストリー

ムデータを無限に続くタプルとしてモデル化しているために，ストリームデータに関係演算を適用するには，それらを関係データに変換する必要があるからである．(2) が必要な理由は，(2) が関係 DBMS で使われる演算処理であることから明らかだろう．(3) が必要な理由は，関係演算により処理された結果を，ストリームとして他のオペレータへ流すために使われるからである．

STREAM において主眼が置かれた研究はコンティニュアルクエリの効率的な実行方式である．実行時間と使用メモリ量においてコンティニュアルクエリを効率的に実行するために，オペレータごとにパイプラインを構成し，それらをスケジューリングする方式を実現している [Babcock *et al.* 03]．

STREAM はコンティニュアルクエリの効率的な実現方式を提供する．しかし STREAM では実時間処理が念頭におかれておらず，データ到着毎のクエリ実行のみしか考えられていないため，多数のコンティニュアルクエリを高精度で周期的に実行することは難しいと考えられる．

3.2.1.2. TelegraphCQ

TelegraphCQ[Chandrasekaran *et al.* 03] は PostgreSQL を拡張して開発された DSMS である．TelegraphCQ の基本的な設計思想は STREAM と同様なため，簡潔に説明する．TelegraphCQ の特徴は動的適応性であり，システム負荷に合わせて異なるクエリ処理方式を用いることができる．

TelegraphCQ のコンティニュアルクエリは STREAM 同様に実時間性を考慮していないために周期的発信の実現が難しいと考えられる．

3.2.1.3. Aurora

Aurora[Carney *et al.* 02] は MIT/Brandeis/Brown 大学で共同開発している DSMS である． ユーザインタフェースには宣言的言語ではなく，オペレータを配置・接続する GUI が提供される．Postgres の生みの親である Michael Stonebraker 博士が開発メンバに入っているが，完全にゼロからスタートしたプロジェクトであり，TelegraphCQ とは関係ない．Aurora の基本的な設計思想は STREAM と同様なため，簡潔に説明する．

Aurora のコンティニュアルクエリは STREAM 同様に実時間性を考慮していないために周期的発信の実行が難しいと考えられる．

3.2.1.4. DSMS のまとめ

DSMS をセンサ応用システムに適用する際の問題は，コンティニュアルクエリは実現されているものの，実時間性が考慮されていない点にある．実時間性が考慮されなければ高精度の周期的発信を実現することが難しいと考えられる．それゆえ，前

述の DSMS に関するいずれの研究においても $Q_{\text{周期的発信}}$ を解決することは難しいと考えられる．

なお，DSMS をそのままセンサ応用システムに適用することは次の理由により困難である．(1)DSMS はデータを永続化しないために $Q_{\text{高鮮度化}}$ を解決しない．(2)DSMS は関係データモデルしか支援しないために $Q_{\text{時系列処理}}$ を解決しない．

3.2.2. 実時間データベースシステム (RTDBMS)

実時間データベースシステム (RTDBMS) の研究の中には，実時間性と時間的一貫性を両方とも扱う研究がある．実時間性とはデッドライン以内に処理を終了させる性質を意味し，時間的一貫性は staleness(s) が一定値以下であることを意味する．本小節では STRIP，StarBase，そして QMF という 3 つの RTDBMS を取り上げる．

3.2.2.1. STRIP

STanford **R**eal-time **I**nformation **P**rocessor(STRIP) は，Stanford University で開発されたソフトリアルタイムデータベースシステムである [Adelberg *et al.* 96]．STRIP は鮮度を保証するために，鮮度が高いセンサデータの到着をトランザクションに待たせるオンデマンドなスケジューリングポリシを採用することで，時間的一貫性と実時間性を高めている．

3.2.2.2. StarBase

StarBase[Lehr *et al.* 95] は，University of Virginia で開発された RTDBMS である．StarBase の特徴は，実時間 OS である RT-Mach を使用することで OS が提供する実時間性を活用している点にある．例えばトランザクション要求は RT-Mach が提供する RT-IPC(Inter Process Communication) を使用し，トランザクションスケジューリングには RT-Mach が提供する実時間スレッドスケジューリングを使用する．

3.2.2.3. QMF

QMF(a **Q**os management architecture for deadline **M**iss ratio and data **F**reshness) は主記憶で動作する RTDBMS のアーキテクチャである [Kang *et al.* 04]．QMF はフィードバック制御，アドミッション制御，そして鮮度を考慮したトランザクションスケジューリングにより実時間性とデータ鮮度を保証するデータベースシステムアーキテクチャであり，シミュレーションにより評価されている．

3.2.2.4. RTDBMS のまとめ

周期的発信は DBMS 内部でのクエリ実行により実現される．RTDBMS は外部から与えられたトランザクションの実時間処理について深い議論がされているものの，

DBMS 内部で実行されるコンティニュアルクエリについて考慮した研究はされていない．それゆえ RTDBMS は $Q_{周期的発信}$ を解決しないと考えられる．

なお，RTDBMS をセンサ応用システムにそのまま適用することは次の理由から困難だと考えられる．(1)RTDBMS は高速データ永続化について考慮していないため $Q_{高鮮度化}$ を解決しないと考えられる．(2)RTDBMS はオブジェクト関係データモデルを考慮していないため $Q_{時系列処理}$ を解決しないと考えられる．

3.2.3.　周期的発信に関する従来研究のまとめ

周期的発信に関する従来研究には DSMS と RTDBMS があった．

DSMS ではコンティニュアルクエリの効率的な実行について研究しているが，その実時間処理については考慮していない．そのために DSMS は高精度でコンティニュアルクエリを実行することは困難だと考えられる．それゆえ DSMS のコンティニュアルクエリ実行方式では $Q_{周期的発信}$ を解決することが困難だと考えられる．

RTDBMS ではトランザクションの実時間処理について研究しているが，DBMS 内部で継続的に実行されるトランザクションについては考慮されていない．それゆえ RTDBMS のトランザクション処理方式では $Q_{周期的発信}$ を解決することが困難だと考えられる．

また，両研究をそのままセンサ応用システムに適用することは次の理由から難しいと考えられる．(1) いずれも高速データ永続化処理を考慮していないため $Q_{高鮮度化}$ を解決することが困難だと考えられること．(2) いずれもセンサデータオブジェクトを扱うデータモデルを考慮していないため $Q_{時系列処理}$ を解決することが困難だと考えられること．

3.3.　時系列処理

本節では $Q_{時系列処理}$ に関する関連研究として，関係データベースシステムの時系列拡張に関する研究と類似シーケンス検索システムを取り上げる．

3.3.1.　関係データベースシステムの時系列拡張

3.3.1.1.　Informix 時系列データブレード

関係 DBMS である Informix は，データブレードというオブジェクト拡張を許す [Informix 97]．データブレードには様々なものがあるが，その中に時系列オブジェクト拡張である時系列データブレードがある．時系列データブレードは主に経済データの管理を目的とし，時系列データとカレンダーデータを管理する．時系列データに対しては秒単位での格納を許し，データを統計解析するための演算として，選択，修正，抽出，交差，結合，入れ換え，統計演算 (減衰関数，移動平均，合

計，単値比較)，数値演算 (四則演算，冪乗，指数，対数，モジュロ，丸め込み，平方根，三角関数)，集約，そして遅延，がある．

Informix 時系列データブレードは上記のような幅広い演算を持つが，類似時系列データ検索演算は提供しない．それゆえ Informix 時系列データブレードは $Q_{\text{時系列処理}}$ を解決しないと考えられる．また，Informix 時系列データブレードをそのままセンサ応用システムに適用することは次の理由から困難だと考えられる．(1) 高速データ永続化処理を考慮していないために $Q_{\text{高鮮度化}}$ を解決しないと考えられること．(2) 連続的なトランザクション実行を考慮していないために $Q_{\text{周期的発信}}$ を解決しないと考えられること．

3.3.2.　類似シーケンス検索システム

類似シーケンス検索技術はパタン認識のために様々なセンサ応用システムで使われている．センサ応用システムは本論文のターゲットであるため，関連研究と考えることに違和感を覚えるかもしれない．しかしながら，データベースシステムは基盤システムであり，基盤システムの使命のひとつにはアプリケーション開発の容易化がある．アプリケーション開発を容易化するために，アプリケーションに実装される必要のある処理を基盤システムが提供すれば，アプリケーション開発者にとって有益だと考えられる．この観点からターゲットアプリケーションを本研究に関連する研究であると見做し，以下に述べる．

3.3.2.1.　ハミング検索

ハミング検索とはメロディの一部を歌唱してその楽曲を検索することである．文献 [小杉 他 04] はハミングによる音楽検索システムである SoundCompass について述べている．SoundCompass は楽曲データを時系列表現してデータベースに格納しておく．この際，音高値の時系列特徴ベクトルと音高差分布特徴ベクトルをデータベース内の楽曲データについて作成しておく．そしてハミングクエリが与えられた際には，ハミングについても特徴ベクトルを生成し，クエリベクトルと楽曲データベクトルについて City-block 距離 [高根 80] を精度基準として検索を行う．実験を通して提案方式の精度は DTW 距離 (第 2.3.3 節参照) を用いた検索方式よりも優れることが示されている．

3.3.2.2.　人–ロボット相互適応メカニズム

文献 [本田 03] では，人間の身体動作とロボットの行動との対応付けを行い，人間とロボットとのコミュニケーションにおける相互適応メカニズム HURMA の実現を光学式モーションキャプチャシステムを用いておこなっている．

人間の身体動作とロボットの行動を対応させるためには，ある時刻における人間の動作と類似する過去の動作を探索する必要があるため，HURMA はモーション

キャプチャから得られたデータを時系列表現し，距離尺度 LCSS[Vlachos *et al.* 03] を用いて類似時系列データの探索をおこなう．距離尺度として LCSS を使った理由を同論文は次のように述べている．「ユークリッド距離は同じ長さの 2 つの時系列データの各点同士との距離を累積していくため，時間的伸縮に対応できない．DTW は異なる長さの時系列データ同士の距離を求めることができるが，1 つの時系列データの各点を比較対象の時系列データのどこかの点に対応させてしまうために，フレーム落ちしたセンサデータが存在する場合に誤差が生じる可能性がある．それらに対して LCSS は類似した点のみを求めることができるため，フレーム落ちを無視して類似度を計算できる」．この研究結果が示すように，欠落のあるセンサデータに対してパタン認識を行うときの距離尺度には LCSS が有効であるかもしれない．

3.3.2.3.　動作認識

文献 [大崎 他 99] では，分割されたデータをクラスタリングして動作の基本パターンを抽出する方法が提案されていると共に，抽出された基本動作パタンに一意に識別できるシンボルを予め与えておいてシンボルを自動的に動作データに付与して動作データの内容を識別する方法が提案されている．この基本パタンのクラスタリングを行う際の距離関数には DTW が使われている．24 個のラジオ体操の動作データから基本動作パタンを抽出した結果，動きの多い部位である手については 80%以上，動きの少ない部位である足については 60%程度の割合で分類に成功している．

3.3.2.4.　類似シーケンス検索システムのまとめ

それぞれの類似シーケンス検索システムは，それぞれが目的とする認識処理を実現するためにふさわしい距離尺度を用いることにより，優れた成果を出している．距離尺度としては City-block 距離，DTW 距離，そして LCSS 距離が使われていた．データベースシステムとしては，これらのセンサ応用システムが行う認識処理を支援するために，様々な距離尺度を用いることが重要であると考えられる．

3.3.3.　時系列処理に関する従来研究のまとめ

時系列処理に関する従来研究には関係データベースシステムの時系列拡張と類似シーケンス検索システムがあった．

関係データベースシステムの時系列拡張に関する研究は，さまざまな統計演算を支援するが，パタン認識を考慮していないために類似シーケンス検索手法を提供しない．それゆえ $Q_{\text{時系列処理}}$ を解決することは難しいと考えられる．

類似シーケンス検索システムは様々なパタン認識を行う．パタン認識処理は何らかの距離関数を用いてシーケンスの類似性を判定する．本論文で取り上げたセンサ応用システムでは City-block 距離，DTW 距離，そして LCSS 距離が使われていた．

ところで，時系列処理に関する従来研究をあえてそのままセンサ応用システムに適用することは困難である．なぜなら，従来研究は周期的発信と高速データ永続化を考慮していないため，$Q_{\text{高鮮度化}}$ と $Q_{\text{周期的発信}}$ を解決しないと考えられるからである．

3.4. 過負荷制御

過負荷制御方式には，トランザクションのインプリサイス処理とアドミッション制御がある．トランザクションの内容には更新と検索の 2 種類があるが，インプリサイス処理はクエリ処理に関する手法 [Vrbsky *et al.* 93, Lin *et al.* 87] のみが考案されており，更新を不完全にする方式は存在しない．アドミッション制御には幅広い手法 [Kang *et al.* 04, Adelberg *et al.* 96, Datta *et al.* 97, Chandrasekaran *et al.* 04] があるが，その中でも更新トランザクションに注目した研究について述べる．この理由は，センサ応用システムにおいては頻繁に到着するセンサデータが大きな負荷の原因となる可能性が高く，センサデータを考慮しない方式よりも本研究との関連が深いからである．

3.4.1. トランザクションのインプリサイス処理

トランザクションが検索処理であり，一定時間内にその検索結果を求められる場合，検索処理結果を段階的に返却することで，徐々に結果を返却する Approximate Query Processing(AQP) と呼ばれる検索処理方式がある [Vrbsky *et al.* 93]．これはリアルタイムシステムにおけるインプリサイス計算モデル [Lin *et al.* 87] をデータベースシステムへ適用した方式である．

AQP における解品質の向上は performance profile と呼ばれる時間関数として表現される．従来の問題解決アルゴリズムでは，すべての処理が終了してから解を出力するため，解品質は図 3.3(a) に示すように変化する．それに対して AQP では図 3.3(b,c) に示すように解品質は時間と共に単調増加する．

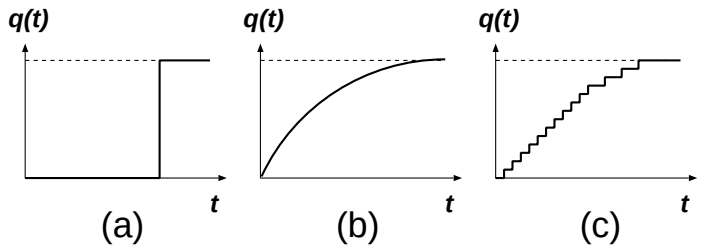


図 3.3: 処理品質の時間変化

検索トランザクションの中間結果取得方式は，過負荷状況においてもある程度の結果を返却できるために，過負荷制御には極めて有効である．しかしセンサデータが過負荷を引き起こすようなシステムにおいては，検索トランザクションの品質が低下し続けてしまう．

3.4.2. トランザクションのアドミッション制御

更新トランザクションに注目した受け入れ拒否方式として Virtual Deadline of Update(VDU) がある．この手法はセンサデータの更新に注目し，アプリケーションが許容できる時間範囲において，システムに到着したセンサデータの処理を拒絶・破棄することによりシステムの負荷を減らす手法である [Datta *et al.* 97]．あるセンサ S について，S に対する更新は前回のアップデート時刻から Virtual Deadline(VD) と呼ばれる時刻まで拒絶される．例えば S の前回の更新時刻が T であり，VD が時刻 $T + A$ だとする．このとき時刻 $T + B(B < A)$ に到着した S への更新は破棄される．

VDU の問題点は更新トランザクションを完全に破棄してしまう点にある．このとき更新トランザクションを完全に破棄するのではなく不確実だが軽量な更新処理手法をトランザクションへ適用できれば好ましい．なぜならその場合にはそれらのセンサデータを永続化できる可能性があるため，将来において閲覧できるかもしれないからである．

3.4.3. 従来研究のまとめ

従来研究には，トランザクションの中間結果取得方式 (**I**mprecise) とトランザクションの受け入れ拒否方式 (**A**dmission) の 2 種類があった．そしてトランザクションにはクエリ (**Q**uery) とアップデート (**U**pdate/Insert) の 2 種類がある．

そうすると過負荷制御を実現するアプローチを表 3.2 に示す 4 種類に分類できると考えられる．この中で (**I**, **Q**) アプローチと (**A**, **Q**) アプローチは第 3.4.1 節で述

表 3.2: 考えられる過負荷制御方式のアプローチ

トランザクション処理の種類	クエリの種類	既存研究の有無
Imprecise	**Q**uery	有
Admission	**Q**uery	有
Imprecise	**U**pdate/Insert	**無**
Admission	**U**pdate/Insert	有

べたように従来研究で扱われている．そして (**A**, **U**) アプローチは第 3.4.2 節で述べたように従来研究で扱われている．しかしながら (**I**, **U**) アプローチは私の知る

限り従来研究では扱われてこなかった．センサデータを扱うデータベースシステムにおいてセンサデータの Update/Insert 処理は高負荷であるため，(**I**, **U**) アプローチは過負荷制御に有効だと考えられる．そこで本研究では第 5 章でこの課題 (**I**, **U**) に挑む．

3.5.　本章のまとめ

本章では第 2 章で定式化した 4 つの問題である $Q_{\text{高鮮度化}}$，$Q_{\text{周期的発信}}$，$Q_{\text{時系列処理}}$，$Q_{\text{過負荷制御}}$ の解決に関する従来研究を述べた．

$Q_{\text{高鮮度化}}$ に関する従来研究は主記憶 WAL と Differential Logging があった．$Q_{\text{周期的発信}}$ に関する従来研究には DSMS と RTDBMS があった．$Q_{\text{時系列処理}}$ に関する従来研究には関係 DBMS の時系列拡張と類似シーケンス検索システムがあった．そしていずれの従来研究も 3 つの課題を同時に解決するものではないことを述べた．$Q_{\text{過負荷制御}}$ に関する従来研究には (**I**, **Q**) アプローチ，(**A**, **Q**) アプローチ，そして (**A**, **U**) アプローチがあるものの，(**I**, **U**) アプローチがまだ研究されていないことを述べた．

以後，本論文は第 4 章において $Q_{\text{高鮮度化}}$・$Q_{\text{周期的発信}}$・$Q_{\text{時系列処理}}$ を同時に解決するデータベースシステム KRAFT を提案し，第 5 章において $Q_{\text{過負荷制御}}$ に対して (**I**, **U**) アプローチから問題解決に挑む手法を提案する．

第4章

センサデータの高鮮度化・周期的発信・時系列処理を実現するデータベースシステム

本章では4つの課題のうち，$Q_{\text{高鮮度化}}$，$Q_{\text{周期的発信}}$，$Q_{\text{時系列処理}}$ を同時に解決するデータベースシステム KRAFT(Kernel of Real-time Active and Fresh Time-series data manager) を提案する．

4.1. 設計指針

4.1.1. 設計方式の議論

KRAFT を実現するにあたり，各問題を解決する機能をデータベースシステムに組み込んでスクラッチから開発する方式 (組み込み方式) と，単純に各機能とデータベースシステムを組み合わせる方式 (ラッパ方式) が考えられるが，組み込み方式はラッパ方式よりも性能が優れると考えられる．ラッパ方式の性能が組み込み方式よりも劣る理由を以下に述べる．

4.1.1.1. リモートロギング法についての比較

ラッパ方式でリモートロギング法を実現するには，センサデータ用のログをリモートロギングシステムで管理し，関係データ用のログをデータベースシステムで管理し，それらを束ねる統合ログマネージャを開発する方法が考えられる．この方法をとるならば，リカバリとチェックポインティング時に統合ログマネージャが両ログを合わせて統合ログを作成し，それを用いてリカバリもしくはチェックポインティングを実行する．

組み込み方式では統合ログを作成する必要はないから，統合コストは存在しない．それゆえリモートロギング法の性能は，ラッパ方式よりも組み込み方式の方が

優れる．

4.1.1.2.　周期的監視機能についての比較

ラッパ方式で周期的監視機能を実現するには，監視処理の呼出毎にデータベースシステムにクエリを発行し，プロセス間通信により結果を取得する必要がある．

組み込み方式ではこの処理は不要なため，周期的監視機能の性能はラッパ方式よりも組み込み方式が優れる．

4.1.1.3.　類似シーケンス検索についての比較

ラッパ方式で類似シーケンス検索を実現すると次のような処理が行われる．(1) ユーザからのクエリをラッパが受け取り，(2) それをパーザが解析して内部表現および SQL クエリに変換し，(3) データベースシステムに SQL クエリを発行してその結果を取得し，(4) ラッパ内で類似検索を実行し，(5) そして最後に処理結果をクライアントに返却する．ここで負荷が大きいのは (3) である．なぜなら (3) ではデータベースからラッパへクエリ結果を転送するからである．このとき結果を転送するには時間がかかる上，転送中には転送データを送信側と受信側で保持するためメモリも必要になる．

組み込み方式では (3) は不要なのでラッパ方式は組み込み方式より多くのメモリと長い処理時間を必要とする．それゆえ類似シーケンス検索の性能は，ラッパ方式よりも組み込み方式が優れる．

4.1.2.　組み込み方式を実現する上の課題

以上の議論より，組み込み方式はラッパ方式よりも優れた性能を提供することがわかった．組み込み方式の実現はラッパ方式とは異なり，各機能の単純な組み合わせでは実現できない．その実現にはデータベースシステム内の様々なモジュールに修正を施す必要がある．組み込み方式を実現するために必要な課題を以下で述べると共に，各課題の解決への参照を示す．

4.1.2.1.　リモートロギング法についての課題

組み込み方式でリモートロギング法を実現するには，関係データベースシステムのための WAL 処理を，リモートロギング法に修正する必要がある．そのため，ロギング実行処理・リカバリ処理・チェックポインティング処理を修正する必要がある．

これらの解決の内，ログセンダについては第 4.2.2.1.4 節で，トランザクションマネージャについては第 4.2.2.1.6 節で，そしてログサーバについては第 4.2.2.2 節で述べる．リカバリの解決は第 4.2.2.1.5 節で述べる．チェックポインティングの解決は第 4.2.2.2.3 節で述べる．

4.1.2.2.　周期的監視機能についての課題

組み込み方式で周期的監視機能を実現するには，クエリの答えをクライアントに転送する時間によって監視処理の周期をずらされない手法が導入されなければならない．例えば周期が1秒の監視処理において，クエリ結果のクライアントへの転送に2秒かかると，監視処理が周期的に動作できない．それゆえ周期的監視を行うスレッドと結果転送を行うスレッドを作成し，それらをつなぐ機構が必要である．そしてパーザにも修正が必要となる．

この解決は，周期的監視の実現については第4.2.2.1.1節で述べ，パーザの修正は第4.2.2.1.2節で述べる．

4.1.2.3.　類似シーケンス検索についての課題

組み込み方式で類似シーケンス検索を実現するには2つの問題がある．

第1の問題はシーケンスデータを保持するセンサ型を関係データベースシステムに導入する必要があることである．この解決のためには，新たなデータモデルを考案し，関係データベースシステムのメモリ管理とストレージ管理を修正する必要がある．センサ型を導入するデータモデルは第4.2.1.1節で述べる．メモリ管理は第4.2.3.1節で述べる．ストレージ管理は第4.2.3.2節で述べる．第2の問題は，類似シーケンス検索要求を解釈するようパーザを修正し，検索処理をエグゼキュータに含めることである．パーザは第4.2.2.1.2節で，エグゼキュータは第4.2.2.1.3節で述べる．

4.2.　設計方式

本節ではKRAFTのデータモデルおよびアーキテクチャに関する設計を述べる．

4.2.1.　データモデル

ここではKRAFTのデータモデルの設計およびその設計の必然性を述べる．

4.2.1.1.　基本設計

KRAFTでは全てのデータは論理的にテーブルとして表現される．KRAFTが提供するデータ型は，整数，実数，日付，固定長テキスト，センサ整数型，そしてセンサ実数型である．この中で，センサ整数型とセンサ実数型がKRAFTが独自に導入したセンサ型である．これらセンサ型の要素であるセンサデータオブジェクトは，センサデータがデータベースサーバに到着した時刻と，センサデータ値の組からなる．センサ型は複数個のセンサデータオブジェクトを保持する．KRAFTは複数個

のセンサデータオブジェクトをシーケンスとみなし，そこから類似するサブシーケンスを切り出す手段を SQL の拡張として与える．

図 4.1 はロボット実験に用いるテーブル例を示している．属性は整数型の id，日付型の experiment_date，そしてセンサ整数型の sonar_data からなる．メモリマップの実装については第 4.2.3.1 節で述べる．

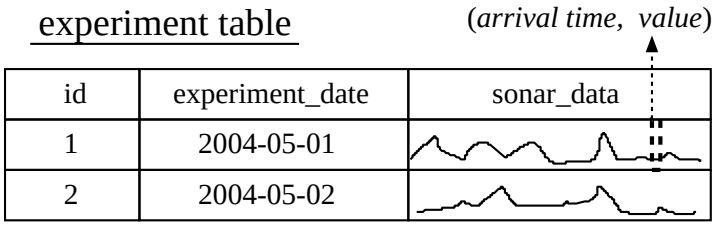


図 4.1: KRAFT のデータモデル

4.2.1.2. 操作インタフェース

表 4.1: KRAFT が提供する操作

操作コマンド	内容詳細
(create|drop) table	Create or drop a table
(create|drop) monitor	Create or drop a monitor
select	Retrieve data from database
insert	Insert a new tuple into table
append	Append a new sensor data into sensor attribute
delete	Delete tuples
update	Update tuples

表 4.1 は KRAFT が提供する操作を示す．このうち，**append** 操作は KRAFT 特有で，新しいセンサデータオブジェクトを追加するために使われる．例えば，experiment というテーブルの中の，id が 1 であるタプルに値が v のセンサデータを追加するには，“append into experiment.sensor_data values (v) where id $=$ 1” と記述する．

select 操作では，選択・射影そして結合をサポートするが，入れ子になった select はサポートしない．結合演算のアルゴリズムは入れ子ループ結合である．

4.2.1.3. データモデルの必然性

データベースシステムのデータモデルは，ユーザにとって使いやすく，さらに優れた性能を提供する必要がある．例えば，KRAFT のユーザである Robovie 実験者は，センサデータを実験日や実験名と関連付けて管理することをデータベースシステムに求める．これを満足するモデルとして，センサデータを表現する属性をテーブル内に入れるデザインは直感的でわかりやすい．これを KRAFT データモデルと呼ぶ．

他方，関係データモデルでデータを表現しようとすると，テーブル ID・時間そしてデータを属性としてもつテーブルと，テーブル ID・実験日そして実験名をもつテーブルを用意しておき，検索時にそれらを結合演算でまとめることで関連付けを行うことになる．センサ毎にテーブルを作ると多くの結合演算を実行する必要があるため処理コストがかかる上，直感的にデータ間の関連を理解しづらい．さらに類似演算の実行はデータベースシステムからセンサデータを受け取った後になるため，関係データモデルでデータを表現した場合には KRAFT データモデルでは不要な通信コストが必要になる．

それゆえユーザにとって使い易い操作インタフェースと優れた性能を提供するためには，関係データモデルでなくて KRAFT データモデルを用いることは必然であった．

4.2.2. アーキテクチャ

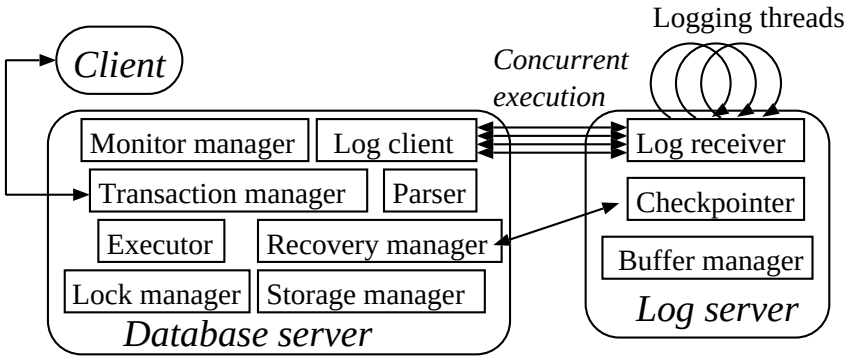


図 4.2: KRAFT のアーキテクチャ

図 4.2 は KRAFT のアーキテクチャ設計を示す．KRAFT は 2 つのサーバから構成される．それらはデータベースサーバとログサーバである．データベースサーバはクエリをクライアントから受け取り，それを処理して，結果をクライアントに返す．ログサーバは通常のローカル WAL ファイルの代わりにログレコードを管理し，データベースサーバと通信する．この実現にはリモートロギング法を適用している

が，データベースシステムに導入するにあたり，複雑なデータ構造の適用と冪等なリカバリ処理[1]を新たに実現した．このアルゴリズムは第 4.3.1.2 節で述べた．

4.2.2.1.　データベースサーバのモジュール構成

ここではデータベースサーバが持つモジュールを説明する．

4.2.2.1.1.　モニタマネージャ　組み込み方式で周期的監視機能を実現するにはモニタマネージャが必要である．しかし既存のデータベースシステムにこのモジュールは存在しないため，私はこれを新たに開発した．

このモジュールは各モニタの周期とクエリ内容を保持すると共にその実行も管理する．モニタの実行は 2 つのスレッドによるパイプライン方式で実行される．一方のスレッドは監視結果を作成し，それをスレッド内部のキューに挿入する．もう片方のスレッドはキューから結果を取り出してクライアントに送る．この実行機構は，処理を分割することで監視処理の周期がずれることを防ぐよう設計された．

4.2.2.1.2.　パーザ　組み込み方式で周期的監視機構と類似シーケンス検索を実現するには，それらの要求を解釈できるようにパーザが修正される必要がある．

このモジュールは SQL ライクなクエリで発行される類似シーケンス検索・周期的監視の作成と実行そしてセンサデータ追加コマンドである **append** を解釈して内部表現に変換できるように開発された．

4.2.2.1.3.　エグゼキュータ　組み込み方式で類似シーケンス検索を実現するには，エグゼキュータを修正する必要がある．類似シーケンス検索として $\mathsf{Dist}_{\mathsf{euclid}}$ と $\mathsf{Dist}_{\mathsf{dtw}}$ を実現し，さらに不等号および等号を処理する機能を追加した．

4.2.2.1.4.　ログセンダ　組み込み方式でリモートロギング法を実現するには，データベースサーバからリモートログサーバへログレコードを転送するモジュールであるログセンダが必要である．既存のデータベースシステムにこのモジュールは存在しないため，私はこれを新たに開発した．

このモジュールは作成したログレコードを TCP プロトコルに則りログサーバへ転送し ack を受け取る．

4.2.2.1.5.　リカバリマネージャ　組み込み方式でリモートロギング法を実現するには，リモートログサーバと通信してログレコードを受け取りリカバリ処理を実行

[1]任意回数の実行結果と 1 回の実行結果が等しくなる性質を冪等性という．文献 [Gray *et al.* 01b]161 ページ参照．

するモジュールがデータベースサーバ内に必要である．既存のデータベースシステムにこのモジュールは存在しないため，私はこれを新たに開発した．

このモジュールはリカバリ処理を実行する．データベースサーバが起動して初期化を済ませた後，最初に呼び出すのはこのモジュールである．このモジュールはリモートログサーバからログレコードを受け取り，redo 処理を冪等に実行する．冪等な redo 処理は第 4.3.1.2 節で述べた．

4.2.2.1.6.　トランザクションマネージャ　組み込み方式でリモートロギング法を実現するには，トランザクションを管理するモジュールが，クライアントとリモートメモリマネージャの両方に対するコネクションを管理する必要がある．なぜならクライアントがデータベースサーバに接続した時に，それに対応するスレッドがデータベースサーバとリモートメモリサーバの両方で一つずつ起動するからである．

このモジュールはスレッドの情報に加えてクライアントとのコネクションを管理する．クライアントがデータベースサーバに接続すると，このモジュールはその情報を内部リストに登録し，それからリモートログサーバに接続して，リモートログサーバ内で新たなロギングスレッドを作成するように依頼する．クライアントがデータベースアクセスを終了する時には，このモジュールはログサーバに関連するロギングスレッドを消去するように依頼し，そして内部リストから当該クライアントの情報を消去する．

4.2.2.1.7.　ロックマネージャ，ストレージマネージャ　組み込み方式を用いる場合，これらのモジュールを既存のデータベースシステムから修正する必要はない．ロックマネージャは並行実行制御を管理する．並行実行制御プロトコルは 2 相ロックであり，シリアライゼーションレベルは read committed である．ストレージマネージャはデータをローカルディスクに書き込む．このモジュールが呼び出されるのはリカバリおよびチェックポインティングである．

4.2.2.1.8.　バッファマネージャ　バッファマネージャは初期化時にユーザに指定された領域を確保し，システムが必要なときにはそれを割り当てる．現在のアルゴリズムはファーストフィットであり，要求されたサイズを提供する．これを図 4.3 に示す．

インタフェースは Kalloc[2] という関数である．Kalloc は KRAFT において，pthread_create 以外のすべてのメモリ割り当てを行う．

メモリが足りない場合には，Kalloc は NULL を返却する．その場合には，テーブルをスワップアウトすることで領域を確保する．スワップアウトはテーブル単位で実行される．現在参照されていないテーブルはスワップアウトの対象となり，メ

[2] KraftAlloc の略．カーネルのメモリ割り当て関数ではない．

モリが解放される．ここで更新もしくは挿入されたタプルについては，タプルに記述されているファイルオフセットへと書き戻される．

```
 1:  メモリ全体を施錠
 2:  if (メモリ割り当てが初めて) {
 3:      ptr := 先頭領域のアドレス;
 4:  }
 5:  else if (末尾に十分な領域がある){
 6:      ptr := 末尾領域のアドレス;
 7:  }
 8:  else if (途中に十分な領域がある) {
 9:      ptr := 途中領域のアドレス;
10:  }
11:  else {
12:      ptr := NULL;
13:  }
14:  return ptr;
15:  メモリ全体を解錠
```

図 4.3: メモリ割り当てアルゴリズム

データベースシステムはあるデータ領域が不要になれば，Kfree コマンドによりその不要データ領域を解放する．不要領域の解放アルゴリズムを図 4.4 に示すと共に解説する．

データ領域を解放するには，そのデータ領域を管理する管理構造領域のアドレスを取得し，その管理構造領域を双方向リストから外せば良い．管理構造領域とデータ領域は連続して割り当てられているために，管理構造領域の先頭アドレスはデータ領域の先頭アドレスから管理構造領域のサイズを引くことにより得られる．

FreeBSD などの汎用 OS ではこのような管理機構が使われず，管理領域とデータ領域は分離されている．この理由は文献 [氷山 04] によれば，二重 Free によるメモリ破壊を防ぐためだという．KRAFT のメモリ機構である Kalloc と Kfree をコールするのは KRAFT の開発者だけであるため，入念な注意をしてプログラミングをすればそのような問題を防げるだろう．それゆえ KRAFT のメモリ機構は安全性よりも性能を重視する設計となっている．

1: メモリ全体を施錠
2: ptr := 解放データ領域のアドレス − 管理構造領域のサイズ
3: ptr が示す管理構造領域を双方向リストから削除
4: メモリ全体を解錠

図 4.4: メモリ解放アルゴリズム

4.2.2.2. リモートログサーバのモジュール構成

リモートログサーバは3つのモジュールから構成される．それらはバッファマネージャ・ログレシーバ・そしてチェックポインタである．これらのモジュールは既存のデータベースシステムに存在しない上に，KRAFT はセンサ型以外の関係データも扱うため，本モジュールは新たに設計・実装された．

4.2.2.2.1. バッファマネージャ　バッファマネージャは，データベースサーバにおいて述べたメモリ割り当て機構と同じ機構である．バッファが足りない場合には，バッファマネージャはチェックポインタを呼び出す．このモジュールは必要不可欠である．なぜならメモリが足りなくなればロギングができなくなるため，メモリ不足時にはそれを解決する手段を自前で持たなければならないからである．

4.2.2.2.2. ログレシーバ　ログレシーバは，ログクライアントのためにロギングスレッドを作成する．従ってロギングは並行的に実行される．ロギングスレッドはログクライアントからログレコードを受け取って管理する．

4.2.2.2.3. チェックポインタ　チェックポインタはすべてのログレコードを回収し，リカバリマネージャにそれらを渡して，データの永続化をおこなう．すべてのタプルにはファイルオフセットを記述することにより，冪等なリカバリを実現している．メモリ不足を解決するために，このチェックポインタは必要不可欠である．Kalloc 呼び出しの返戻値が NULL であったならば，チェックポインタを呼び出し，メモリ不足を解消する．チェックポインタの動作を図 4.5 に示す．

4.2.2.3. モジュール構成の必然性

4.2.2.3.1. リモートロギング法に関するモジュール　リモートロギング法をデータベースシステムに組み込むために，私がログセンダをデータベースサーバに組み込む一方で，ログレシーバ・チェックポインタ・そしてバッファマネージャをログサーバに組み込んだ理由を述べる．

1: ログシリアライザに排他的ロックをかける
2: チェックポインタスレッド (CPT) を作成
3: CPT は全スレッドに対してバッファの切り替えを指示
4: CPT は全スレッドからログリストを切り取る
5: CPT はデータベースサーバ (DBS) に接続
6: CPT は全ログを整列して送信
7: DBS は受け取ったデータを永続化
8: CPT は DBS からデータ永続化成功を受信後，メモリを解放して終了

図 4.5: チェックポインタのアルゴリズム

リモートロギング法では新しいログをローカルディスクには書かずにリモートホストのメモリに書くため，ログをデータベースサーバからログサーバに転送する必要がある．この転送を実現するためにログセンダはデータベースサーバに組み込まれ，ログレシーバはログサーバに組み込まれる．

チェックポインティングはログデータ領域へのアクセスを禁じつつ，ログレコードから復元したデータをデータベースサーバ上のディスクに書き込む処理であるから，チェックポインタはログデータ領域をもつログサーバに組み込まれることが性能面から見て好ましい．

ログサーバはメモリが不足するとログを置けなくなるため，メモリ管理を OS に委ねるわけにはいかない．すなわちログサーバが使うメモリはログサーバが管理し，不足時にはチェックポインタを呼び出す必要があるため，ログサーバにバッファマネージャが組み込まれる．

それゆえ，私がログセンダをデータベースサーバに組み込み，ログレシーバ・チェックポインタ・そしてバッファマネージャをログサーバに組み込んだことは，リモートロギング法を組み込み方式で実現するために必然的だった．

4.2.2.3.2.　周期的監視機能に関するモジュール　周期的監視機能をデータベースシステムに組み込むために，私がモニタマネージャをデータベースシステムに組み込んだ理由を述べる．

モニタマネージャをデータベースシステム内に配置せずとも周期的監視は実現できる．しかし配置がデータベースシステムから遠ざかるにつれて監視処理の性能は劣化し，監視周期の間隔は不正確になる可能性が高まる．そこでエグゼキュータを呼び出せる程度の深度にモニタマネージャを配置することで，一般的なクエリに対応すると同時に性能向上を図った．

それゆえ KRAFT がモニタマネージャをデータベースシステム内部に配置したことは，性能とクエリの多様さを両立する周期的監視を実現するために必然的だった．

4.2.2.3.3.　類似シーケンス検索に関するモジュール　類似シーケンス検索機能をデータベースシステムに組み込むために，私がパーザとエグゼキュータを修正した理由を述べる．パーザはユーザから受け取ったクエリを内部表現に変換するモジュールだから，従来のテーブル操作クエリだけでなく，第 4.5.1 節で述べた類似シーケンス検索クエリも解釈する必要がある．その要求を実行する類似検索実行部をエグゼキュータ外部のモジュールとして開発することも不可能ではないが，エグゼキュータに組み込む方が設計が単純で美しくなり，エグゼキュータと外部モジュール間のインタラクション消去による性能改善も見込めるため，私は類似検索実行部をエグゼキュータに組み込んだ．

それゆえ私が類似シーケンス検索機能をデータベースシステムに組み込むためにパーザとエグゼキュータを修正したことは必然的だった．

4.2.3.　データ配置方式

4.2.3.1.　データベースサーバのメモリマップ

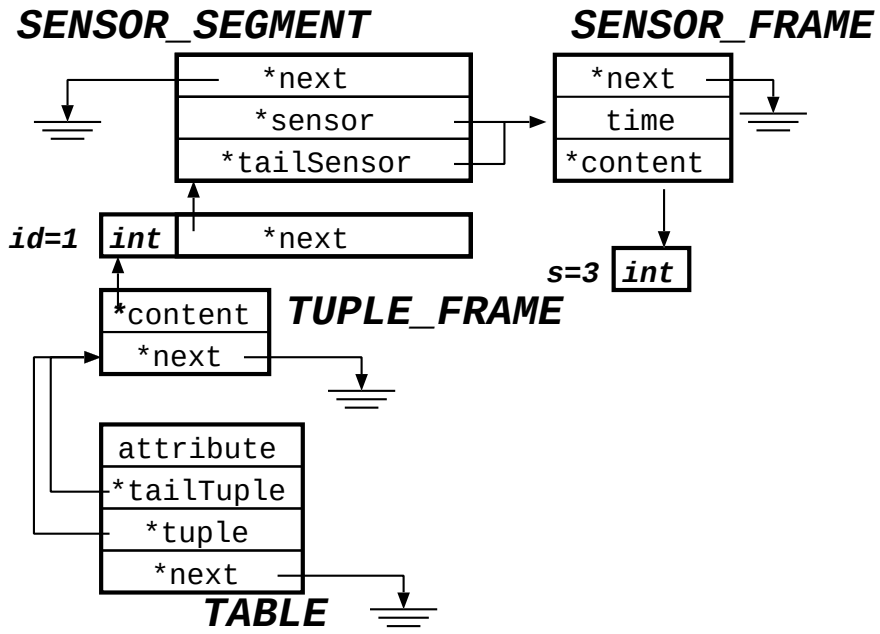
データベースサーバのメモリマップは，通常のデータベースシステムとは若干異なる．なぜなら KRAFT はセンサ型をもつからである．タプルはリストで実現され，データエリアはタプルのサイズに応じて動的に確保される．センサ型がタプルに存在する場合には，センサ型のポインタがタプルエリアに割り当てられる．そして各センサデータオブジェクトはリストにより実現される．

メモリマップの例とそれを作成するコマンドを図 4.6 に示す．この中で，*TABLE* は *TUPLE_FRAME* をさしている．*TUPLE_FRAME* 中の ∗content が指すのは，タプルのデータを保持する領域である，タプルデータオブジェクトである．図 4.6 の場合，タプルデータオブジェクトは INT 型データオブジェクトと，*SENSOR_SEGMENT* 型ポインタオブジェクトから構成される．この INT 型データオブジェクトの値は 1 であり，*SENSOR_SEGMENT* 型ポインタオブジェクトの ∗next は，動的に確保される *SENSOR_SEGMENT* 型データオブジェクトを指す．*SENSOR_FRAME* 型ポインタオブジェクトの ∗sensor は，動的に確保される *SENSOR_FRAME* 型データオブジェクトを指す．*SENSOR_FRAME* 型データオブジェクト内の ∗content はセンサデータオブジェクトを指す．この場合，その型は INT で値は 3 である．

4.2.3.2.　データベースサーバのストレージ構成

図 4.7 は図 4.6 中に記述されたコマンドを用いたときに作成されるテーブルのストレージ構成を表す．これは図 4.1 に示したテーブルスキーマと等しい．

ディレクトリ “/dataroot/experiment/” は 2 つのファイル，“garbage” と “relation”，およびディレクトリ “sensor/” をもつ．“relation” はセンサデータ以外のタプルデータをもち，“garbage” はすでに消去されたタプルのファイルオフセットをも




```
% create table experiment(id int, sonar_data sensor_int)
% insert into experiment values (1, null)
% append into experiment.sonar_data values (3) where id = 1
```

図 4.6: KRAFT のメモリマップ

つ．データベースサーバがローカルディスクからデータを読み込むとき，“garbage” に記述されたファイルオフセットは読み込まれず無視される．

ディレクトリ “/dataroot/experiment/sensor/” は “sonar_data/” というディレクトリをもつ．そしてこのディレクトリには “garbage” と “object” というファイルがある．“object” ファイルはセンサデータオブジェクトを保持し，“garbage” ファイルはすでに消去されたセンサタプルのオフセットをもつ．

ファイル “/tableroot/experiment” は “experiment” テーブル内の属性情報を含む．

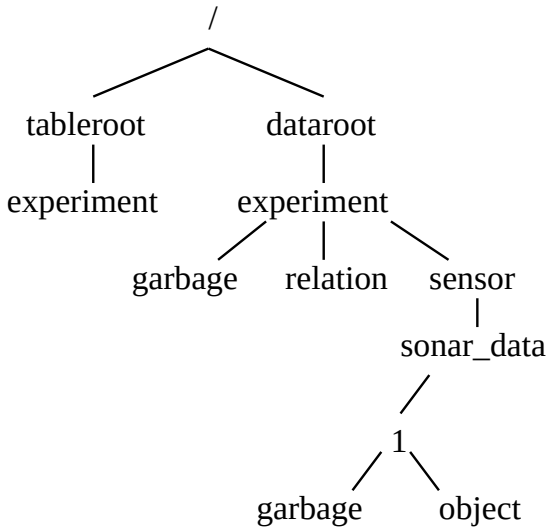


図 4.7: KRAFT のストレージ構成

4.2.3.3. データ格納モデル

図4.7 において，図4.6 中に記述されたコマンドを用いると，非センサデータである id は “/data/root/experiment/relation” に格納され，センサデータである sonar_data は “/data/root/experiment/sensor/sonar_data/object” に格納される．両ファイルのデータ格納モデルは図4.8 に示すように，NSM(N-ary Storage Model) ではあるが，現在，KRAFT のディスク I/O はページ単位ではなくタプル単位であるため，今後，ページ単位のディスク I/O を実現する予定である．

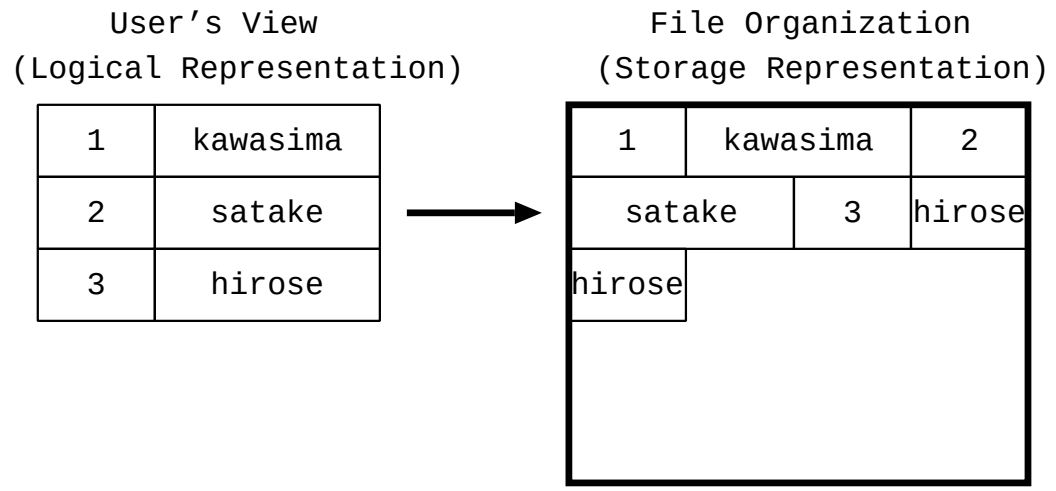


図 4.8: KRAFT のデータ格納モデル

4.3. センサデータの高鮮度化方式

4.3.1. 設計

センサデータの高鮮度化を実現するために，私はリモートロギング法を KRAFT に適用する．一般的に，センサデータに優れた鮮度を与えることに対する最大の障壁は，遅いディスクアクセスである．この問題を解決するために，リモートロギング法ではリモートメモリにログレコードを置く．

リモートロギング法の利点は，全てのセンサデータ追加トランザクションを同時に実行できる点にある．なぜならトランザクション毎にリモートログサーバへコネクションを張り，各トランザクション毎にログを転送するからである．これを図 4.9 に示す．

これによりデータ追加処理を高速化できるため，センサデータを高鮮度化できる．

通常の WAL の問題点は，ログファイルが全トランザクションにより共有されるために，あるトランザクションがログファイルに書き込みを行っているときには，他のトランザクションは待たされることである．これを図 4.10 に示す．

永続性を弱めることで過負荷を制御する方式については第 5 章で述べる．

4.3.1.1. ロギングプロトコル

タプル追加処理 (insert) とセンサデータオブジェクト追加処理 (append) のためのロギングプロトコルを図 4.11 に示す．他の操作コマンドはスペースのために省略する．2 行目において，ログサーバは新しいロギングスレッドを作成する．クライアント一つにつき一つのログスレッドが対応するので，クライアントとログスレッ

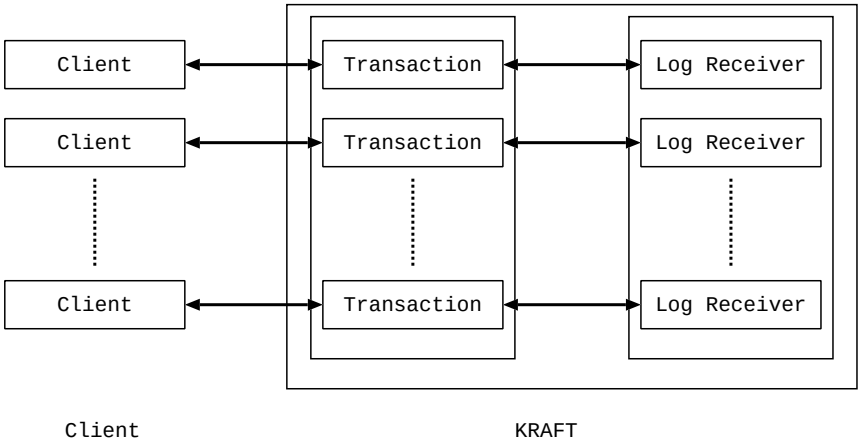


図 4.9: リモートロギング法の利点

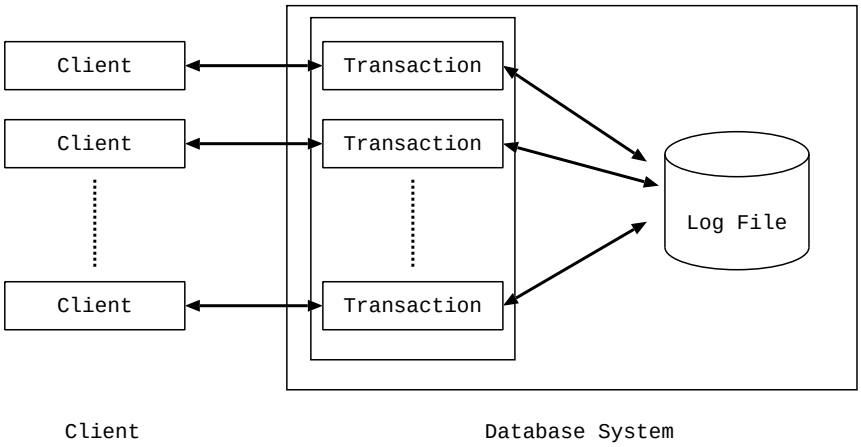


図 4.10: 通常 WAL の問題点

ドの数は等しくなる．

6–11 行目では，タプル追加 (insert) 処理が記述されている．6 行目では，ログサーバはタプルフレームデータ構造 (図 4.6 における “*TUPLE_FRAME*”) とタプルデータを受け取る．なぜならタプルデータ領域は動的に確保されるので，別々に送信される必要があるからである．

9–10 行目では，ログサーバはファイルの末尾オフセットを到着したタプルに与えたあと，そのファイル末尾オフセットを次のタプルに備えて増加させる．この処理は冪等なリカバリ処理を実現するために行われる．リカバリ時には各タプルに記録されているファイルオフセットを利用してリカバリ処理が行われる．

12–17 行目では，センサ追加 (append) 処理が記述されている．これは 6–11 行目と同じ処理である．

1: C connects D, and D connects L.
2: L creates a logging thread for the connection.
3: C sends an operation to D.
4: D sends a log record to L.
5: **switch** (type of the log record) {
6: **case** insert:
7: D sends tuple frame to L and L sends ack.
8: D sends tuple content to L and L sends ack.
9: L gives offset of file tail for the tuple.
10: L increments the offset for the tuple.
11: **break**;
12: **case** append:
13: D sends sensor frame to L and L sends ack.
14: D sends sensor content to L and L sends ack.
15: L gives offset of tail for the sensor.
16: L increments the offset for the sensor.
17: **break**;
18: }
19: L sends ack to D.
20: D sends message to C.

C: client, D: database server, L: log server

図 4.11: ロギングプロトコル

4.3.1.2. リカバリプロトコル

1: D starts and connects to L with recovery port.
2: L collects all of log records limiting in
committed transactions from log threads.
3: L sorts the records by ascending time order.
4: L sends all of the sorted records to D.
5: D makes transactions from the records.
6: D executes redo of the transactions.

D: database server, L: log server

図 4.12: リカバリプロトコル

図 4.12 はリカバリプロトコルを示す．リカバリ処理は並行的には実行されない．従ってロギングよりも時間を要する．2 行目で，L はコミットされていないトランザクションのログレコードを破棄する．これらのレコードは各ロギングスレッドが保持するログレコードの末尾部分に存在する可能性がある．なぜならトランザクションは begin で始まり，commit で終了するからである．5 行目において，D はレコードからトランザクションを再構築する．

4.3.2. センサデータの高鮮度化に関する評価

実験環境として，スイッチングハブで結合された 3 台のホストを用いた．ネットワークの帯域は 100MB イーサネットである．ホストはそれぞれデータベースサーバ，ログサーバ，そしてクライアントに用いた．データベースサーバマシンの仕様は，CPU がクロック周波数 3GHz の Pentium 4，メモリ 4GB，そして OS が Linux Kernel-2.4.21-4 である．ログサーバマシンの仕様は，CPU がクロック周波数 2.4GHz の Pentium 4，メモリ 1GB，そして OS が Linux Kernel-2.4.21-4 である．クライアントマシンの仕様は，CPU がクロック周波数 2.0GHz の Pentium 4，メモリ 1GB，そして OS が Linux Kernel-2.4.21-4 である．各ホストにおいて PTHREAD_MAX_THREADS は 256 に設定されていたため，それ以上のスレッドは同時に作成できなかった．

4.3.2.1. staleness(s) の測定

私は Robovie データの staleness(s) を表 4.2 に示す条件で測定した．まず私は 11 のモータから得られるセンサ値と 24 のソナーから得られるセンサ値を格納する，Robovie という名前のテーブルを KRAFT 内に作成した．そして次に周期的監視を作成した．それから，Robovie.motor1 にセンサデータを追加するアペンダスレッド

を 1 つだけ作成して，周期的監視が得た staleness(s) を測定した．周期的監視の作成に用いたコマンドを図 4.13 に，アペンダの実行に用いたコマンドの疑似コードを図 4.14 に示す．

```
% create table Robovie (id int, motor1 sensor_double, ..., motor11 sensor_double, sonar1 sensor_double, ..., sonar24 sensor_double)
% insert into Robovie values (1, null, ..., null)
% create monitor monitor_Robovie period 1s as select * from Robovie where latest Robovie.motor1
% run monitor_Robovie during 300s
```

図 4.13: 周期的監視の実行に用いたコマンド

```
while (1) {
    append into Robovie.motor1 values (1) where id = 1;
    sleep(period of appender);
}
```

図 4.14: アペンダの実行に用いたコマンド

表 4.2: staleness(s) 測定実験の条件

Number of appender	1
Period of appender	10ms or 50ms
Period of monitor	1s
Duration of monitor	300s

図 4.15 は staleness(s) を測定した結果を示す．この図の横軸は周期的監視処理の試行回数を表し，縦軸は staleness(s) を示す．アペンダの周期は 10ms であるから，10ms 以下の staleness(s) は，センサデータの永続化に遅延をまったく及ぼさないことになる．α，β，γ はそれぞれ 0.8，20ms，1ms と設定した．

図 4.15 において，試行を重ねるにつれて staleness(s) は徐々に上がって行き，10ms を超えると殆んど 0ms まで低下する．その後はまた 10ms へ近付いていくと考えられる．私がこのように考えた理由は，アペンダ周期を 50ms にした場合に図 4.16 の

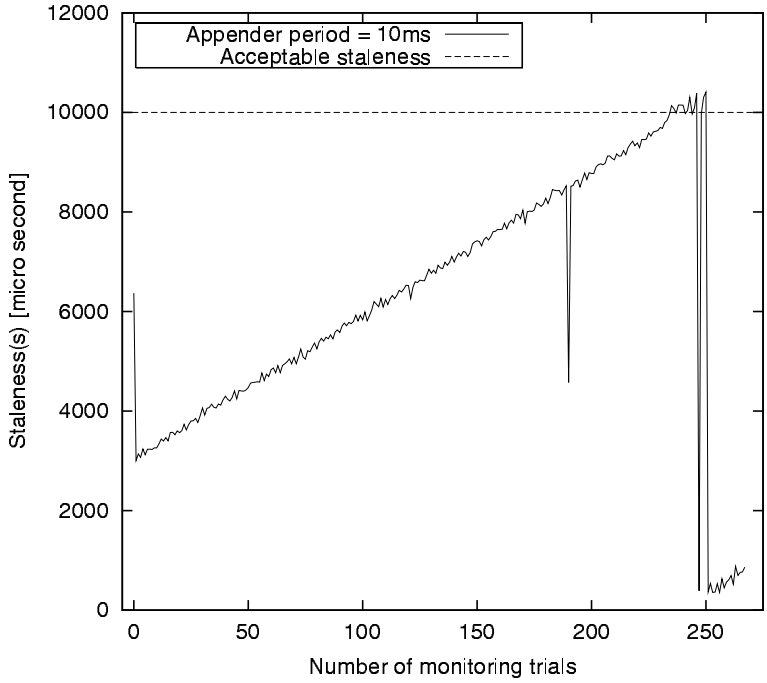


図 4.15: staleness(s) の遷移過程 (アペンダ周期=10ms)

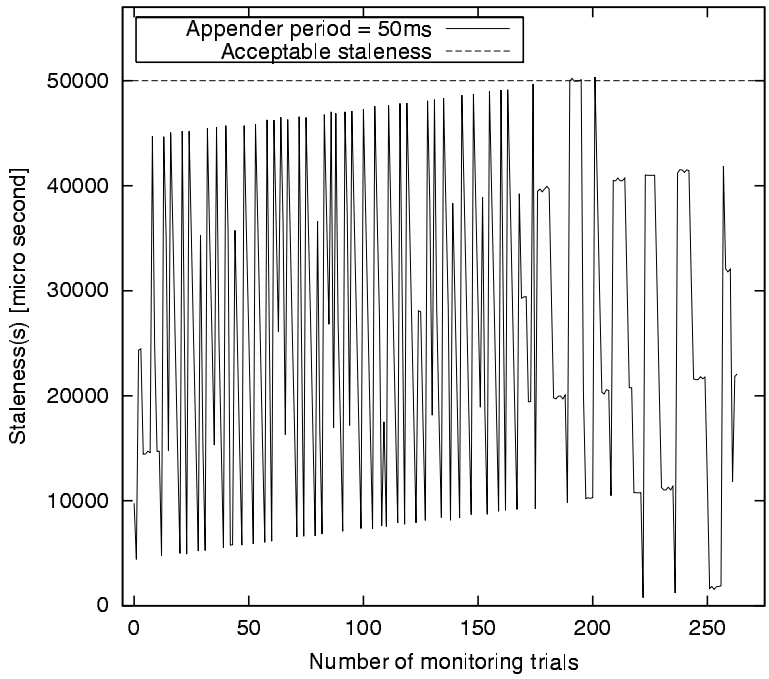


図 4.16: staleness(s) の遷移過程 (アペンダ周期=50ms)

ような結果が得られたからである．図 4.16 からは staleness(s) が 50ms へ近付くと 0ms 近辺に低下する繰り返しが観察される．

staleness(s) が徐々に変動する理由には次の 2 つが考えられる．

1. データ挿入処理と周期的発信の動作機構の違い

   データ挿入処理過程は次のようになる．(1) センサクライアントは sleep から目覚める．(2) センサクライアントはセンサ値を読み取り，それを用いてセンサデータを作成する．(3) そのデータが KRAFT データベースサーバに到着する．(4) リモートログが実行され，データはバッファ領域におかれる．(5) 最後に ack がデータベースサーバからセンサクライアントへ返される．(6) センサクライアントはそこで周期だけ sleep する．つまり (2)〜(5) に要する時間が変動のずれになりうる．このおよその時間を計算する．追加時間を表す図 4.18 において，32 並行度での 100 回の挿入時間は約 330ms だった．従って 1 並行度での 1 回の挿入時間は 300μs 程度になり，これが (2)〜(5) に要する時間となる．以上よりセンサのスリープ周期が 10ms の場合，センサデータが KRAFT に到着する周期はおよそ 10.3ms になっていると考えられる．一方，周期的監視は第 4.2.2.1.1 節で述べたパイプライン機構により，データ読み取り処理自体は変動なく周期的に実行される．それゆえ両者の動きの違いが変動ずれを生まれた可能性がある．

2. センサ発生ホストと KRAFT ホストの時刻ずれ

   実験前に “ntpdate” コマンドを使って同期させたが，センサ発生ホストと KRAFT ホストの時刻が若干ずれていたことが，staleness(s) が徐々に変動したことに影響を与えた可能性がある．

図 4.15 において，周期である 10ms を超えた staleness(s) は全体の 4.10%存在した．定義より staleness(s) が小さいほど freshness(s) は優れるため，この実験結果は KRAFT が優れた freshness(s) を提供することを示している．より詳細な実験結果を表 4.3 に示す．表 4.3 から，staleness(s) の平均値は，ほぼ周期の半分となっており，さらに周期越え率は最悪でも 4.10%であることがわかる．

staleness(s) の平均値がセンサ周期よりも低ければ，最新のセンサデータを読めていたことになるため，平均値がセンサ周期の半分となることは，persistent(s) を得るためのコストを殆んどかけずに，最新センサデータを提供できたことを意味する．平均値が周期の半分になった理由は，図 4.15 より，staleness(s) が徐々に変動するからだと考えられる．

以上より私は KRAFT が極めて小さい staleness(s) を提供すると主張する．それゆえ KRAFT は $Q_{\text{高鮮度化}}$ を解決すると私は主張する．

表 4.3: staleness(s) に関する統計的結果

センサ周期 (ms)	周期越え率 (%)	最悪値 (ms)	平均値 (ms)
10	4.10	10.4	6.23
20	2.25	23.9	11.2
30	1.13	30.4	16.1
40	0.38	94.1	21.6
50	1.13	50.4	26.3
60	0.00	58.2	29.7
70	1.50	76.4	35.7
80	1.12	80.4	43.0
90	0.37	90.6	44.7
100	0.00	98.6	50.3

1: アペンド数を X とする.
2: スレッドの ID を myid とする
3: for ($i = 0$; $i <$ X; i ++) {
4:　　append into X.myid values (i) where id = 1;
5: }

図 4.17: センサデータ追加プログラム

4.3.2.2. センサデータ追加時間の測定

staleness(s) に関する実験で KRAFT が優れたパフォーマンスを示した理由を理解するために，私はセンサデータ追加処理時間を測定した．この実験では，私は多数のアペンダスレッドを作成して並行的に実行させた．各スレッドには一定数のセンサデータ追加処理を発行させ，その合計実行時間を測定した．これをプログラムで示すと図 4.17 のようになる．実験では，X の値を 100 から 500 までに設定した．最大並行度を表す myid の最大値は 32 から 224 まで測定した．

図 4.18 は実験結果を示す．図 4.18 の中で，X 軸はスレッド一つあたりの総データ追加処理数を表し，Y 軸は全スレッドが終了するまでにかかった時間を表す．KRAFT が優れた並行性を示すことを理解しやすくするために，並行度が 32 の場合に対して，何倍の時間を要したかを示す結果を図 4.19 に示す．図 4.19 は，並行度が N 倍になっても実行時間が N 倍より低くなっていることを明らかに示している．それゆえ KRAFT は優れた並行性を示しているといえる．

図 4.19 には，2 つの興味深い点が存在する．それらは，追加処理数が 100 で並行度が 128 の場合と，追加処理数が 100 で並行度が 224 の場合である．ここで性能が悪化している理由は，*pthread_create* もしくはデーモンプロセスの影響を受けたことではないかと私は推測している．

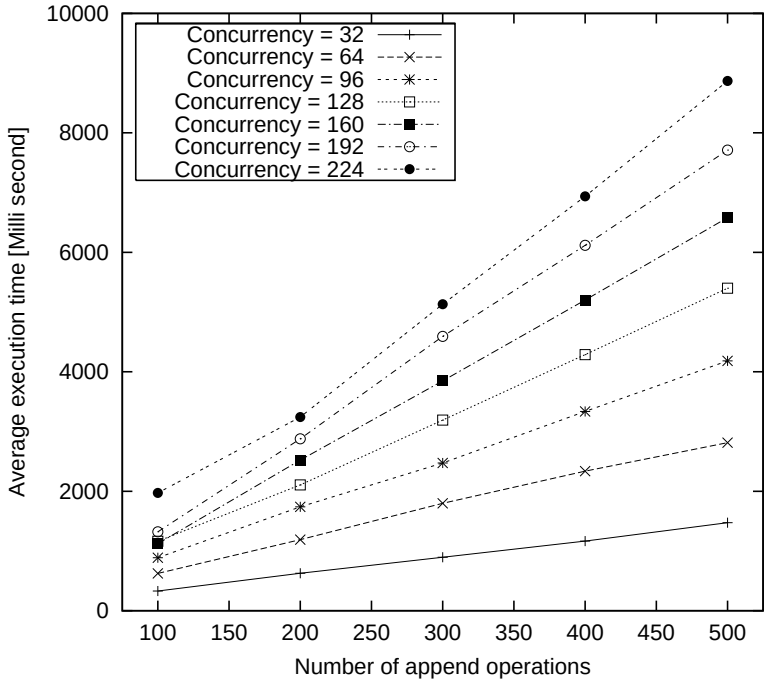


図 4.18: 平均実行時間

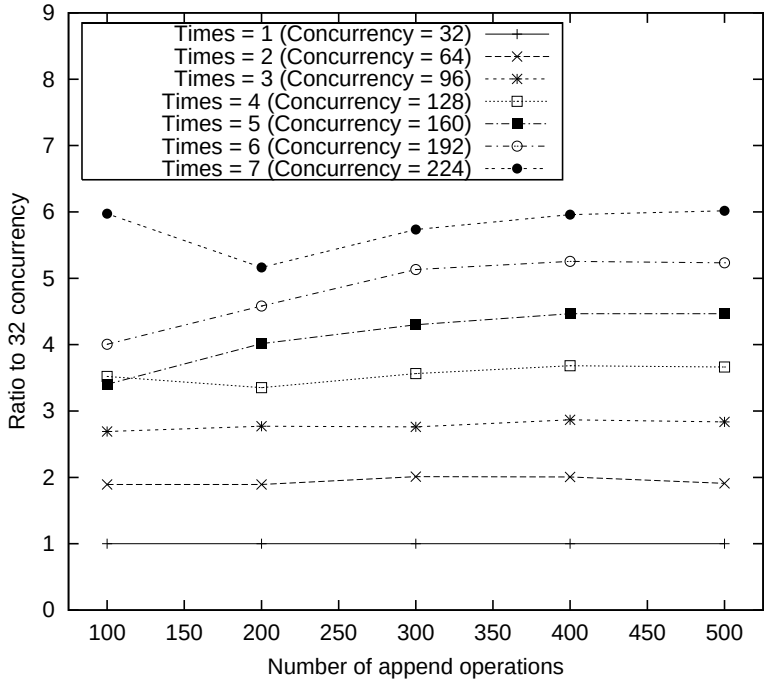


図 4.19: 32 並行度との比較

4.3.3.　まとめ

本節では$Q_{\text{高鮮度化}}$を解決するために，リモートロギング法を組み込み方式で実現したデータベースシステム KRAFT において実験的に評価し，鮮度については平均値がセンサ発生周期の半分程度になり，追加処理時間には優れた並行性が示されることを明らかにした．

4.4.　センサデータの周期的発信方式

4.4.1.　設計

周期的発信を効率的に実行するために，エグゼキュータを呼び出す処理と，エグゼキュータが得た結果を転送する処理を異なるスレッドで実行する．以後，前者をモニタスレッド，後者を転送スレッドと表記する．

モニタスレッドと転送スレッドはパイプライン処理することにより効率的な周期的発信を実現する．両者の機能が同一スレッドで実現された場合の問題は，転送に要する時間がわからないことである．エグゼキュータの結果を転送するのに要する時間は，クライアントへのネットワーク的距離とデータ量に依存するが，基本的には長くかかり，転送時間が監視をおこなう周期を越える可能性も考えられる．そこ

で両者の機能を分割して実現することにより，モニタリングスレッドの負荷を軽減し，周期的なエグゼキュータ実行を実現する．

モニタスレッドと転送スレッドの設計方式を以下で述べる．

4.4.1.1. モニタスレッド

周期的にエグゼキュータを呼び出すモニタスレッドのアルゴリズムを図4.20に示す．

1行目でスケジューラに自分のリリースタイムを通知する．スケジューラはこれによりスレッドのリリースタイムを知り，この情報をもとにスケジューリングを行う．2行目から4行目では，リリースタイムまでの待機を行う．スケジューラがモニタスレッドをリリースタイム前に実行スレッドとして選択した場合，そのモニタスレッドはスケジューラを呼び出し，スケジューラに再度，実行スレッドの選択を行わせる．リリースタイムが過ぎた後，5行目においてモニタスレッドはエグゼキュータを呼び出し，その結果を6行目にあるように結果リストに追加する．当然ながらこのときリストを施錠/解錠する．

```
1:  スケジューラにリリースタイムを通知
2:  do {
3:      スケジューラの呼出
4:  } while(現時刻 < リリースタイム)
5:  エグゼキュータを呼び出す
6:  エグゼキュータの結果を結果リストに追加
```

図 4.20: モニタスレッドのアルゴリズム

4.4.1.2. 転送スレッド

モニタスレッドが得た結果をクライアントに転送する，転送スレッドのアルゴリズムを図4.21に示す．

2行目から4行目において，結果リストに答えがあるかチェックし，もしあればそれをクライアントに転送する．そして結果があろうとなかろうと，5行目において，次に動かすスレッドを選択させるためにスケジューラを呼び出す．

4.4.1.3. スレッドスケジューラ

走らせるスレッドを選択するモジュールである，スレッドスケジューラのアルゴリズムを図4.22に示す．1〜3行目に示されている通り，スケジューラを呼び出したのが転送スレッドであれば，スケジューラはそのスレッドを実行キューの末尾に追

```
1:    while (1) {
2:        if (結果リストに答えがある) {
3:            答えをクライアントに転送
4:        }
5:        スケジューラの呼出
6:    }
```

図 4.21: 転送スレッドのアルゴリズム

加する．この処理に要する計算量はO(1)である．なぜならばこの処理に必要な操作は，実行キューの末尾オブジェクトの次要素への当該スレッドの追加のみであるからである．

4〜6行目に示されている通り，スケジューラを呼び出したのがDBサーバスレッドであれば，スケジューラはそのスレッドを実行キューの先頭に挿入する．なぜならDBサーバスレッドが最優先されなければ，データベースシステムは新しいクライアントからのコネクション要求に応答できないからである．DBサーバスレッドはクライアントからのコネクション要求がなければスケジューラを呼び出すことによりモニタスレッドや転送スレッドを実行させる．この処理に要する計算量はO(1)である．なぜならばこの処理に必要な操作は，実行キューの先頭への当該スレッドの挿入のみであるからである．

7〜9行目に示されている通り，スケジューラを呼び出したのがモニタスレッドであれば，スケジューラはそのスレッドを実行キューの先頭から，その中のモニタスレッドに対してリリースタイムを比較していき，実行キュー内でモニタスレッドがリリースタイムの早い順に並ぶような位置へ挿入する．すなわち，最もリリースタイムが早いものが実行キューの先頭に並べられる．この方式を **ERTF**(Earliest Release Time First) と表記する．この処理に要する計算量は，モニタスレッドの数をNとするとO(N+1)である．なぜならば，この処理に必要な操作は，最も計算量がかかる場合において，実行キュー内の全てのモニタスレッドと比較する必要があるからであり，実行キューの先頭はDBサーバスレッドであり，その後にはN個のモニタスレッドが並んでいるからである．

4.4.2. 実装

4.4.2.1. 開発方法

スケジューラはFreeBSD 5.3 Release(5.3R) における Pthread スケジューラに修正を施して実装した．すなわち同OSソースコードにおいて”/usr/src/lib/libpthread”以下を修正した．FreeBSD5.3Releaseより，-pthreadにより選択されるスレッドラ

```
1:  if (スケジューラを呼び出したのは転送スレッド) {
2:      実行キューの末尾に追加
3:  }
4:  else if (スケジューラを呼び出したのは DB サーバスレッド) {
5:      実行キューの先頭に挿入
6:  }
7:  else if (スケジューラを呼び出したのはモニタスレッド) {
8:      そのスレッドのリリースタイムに合わせてキューに追加
9:  }
```

図 4.22: スケジューラのアルゴリズム

イブラリが libc_r ではなく kse になったため作業手順は次の通りになる．

```
1:  ソースコードを編集
2:  make && sudo make install
3:  gcc -pthread ... として KRAFT を make
```

図 4.23: スケジューラ開発手順

FreeBSD5 系からは，効率的なスケジューリングを実現するために Scheduler Activation(SA) [Anderson *et al.* 91] ベースの設計である，Kernel Scheduling Entity(KSE) が使われている．これは基本的に SA と同様であり，カーネルからメールボックスを通じてユーザレベルへの通知を行う．libc_r と異なり，マルチプロセッサアーキテクチャならばスレッドを異なるプロセッサに割り振ることができる．ユーザスレッドのスケジューリングはユーザレベルスケジューラ (UTS) により実現される．

4.4.2.1.1.　スケジューラの呼び出し方法　クライアントプログラムが UTS を呼び出すには，FreeBSD5.3Release の pthread 実装では pthread_yield をコールすればよい．pthread_yield コールから UTS までの関数遷移を図 4.24 に示す．

4.4.2.1.2.　リリースタイムの設定　モニタスレッドがリリースタイムを設定するために，pthread_set_release(long long release) というインタフェースを作成した．スレッドの構造は struct pthread により定義されており，struct pthread の中には struct pthread_attr 型である attr という名前の属性が存在する．この属性に long long 型で release という名前の属性を追加した．

```
1:    pthread_yield (クライアントプログラムが実行)
2:    _thr_sched_switch
3:    _thr_sched_switch_unlocked
4:    _thread_enter_uts (ユーザレベルスケジューラへの遷移)
5:    _i386_enter_uts (CPU が Intel である場合)
6:    kse_sched_multi(スケジューラ本体)
```

図 4.24: yield によるスケジューラ呼び出し

```
1:    kse_sched_multi
2:    kse_switchout_thread
3:    if (スレッドは走行状態である (PS_RUNNING)) {
4:        _pq_schedule_ertf (筆者により追加された関数)
5:    }
```

図 4.25: 実行キューの操作

pthread_set_release が呼び出されると，スケジューラを施錠し，ユーザが設定した release を属性に代入し，スケジューラを解錠する．そして UTS はこの値をみてそのスレッドをスケジューリングする．

4.4.2.1.3.　実行キュー操作　前述のアルゴリズムを実行するにあたり，優先度キューを複数もつ必要はない．kse->sched_queue->sq_runq->pq_lists; が実行キューであり，この数は THR_MAX_PRIORITY - THR_MIN_PRIORITY + 1 で設定される．そこで，THR_MIN_PRIORITY も THR_MAX_PRIORITY も 0 に設定することで，実行キューを 1 つに減らした．

実行キューの操作はスケジューラである関数である kse_sched_multi から図 4.25 のように呼び出される．

そして _pq_schedule_ertf の実装を図 4.26 に示す．

4.4.2.1.4.　ユーザレベル実装の利点と欠点　実装をユーザレベルでおこないカーネルを修正しない利点は，インストールが容易なことであるが，欠点はカーネル修正に比べれば性能が落ちると考えられることである．一方，カーネルを修正する利点は性能向上だが，欠点はインストールのためにカーネル再構築が必要なために開発に手間を要する事と，バグ混入によりシステム安定性が失われることである．

研究の第一段階としては Pthread レベルのみでの実装によるインストールの容易

```
 1: extern void
 2: _pq_schedule_ertf(pq_queue_t *pq, pthread_t pthread)
 3: {
 4:     int prio = THR_MIN_PRIORITY;
 5:     pthread_t work;
 6:
 7:     PQ_SET_ACTIVE(pq);
 8:
 9:    work = TAILQ_FIRST(&(pq->pq_lists[prio].pl_head));
10:    while (1) {
11:        if ((work == NULL) ||
12:                (work->attr.release < 0) ||
13:                (work->attr.release > pthread->attr.release)) {
14:            break;
15:        }
16:        work = TAILQ_NEXT(work, pqe);
17:    }
18:    if (work == NULL) {
19:        TAILQ_INSERT_TAIL(&(pq->pq_lists[prio].pl_head), pthread, pqe);
20:    }
21:    else {
22:        TAILQ_INSERT_BEFORE(work, pthread, pqe);
23:    }
24:
25:    if (pq->pq_lists[prio].pl_queued == 0) {
26:        pq_insert_prio_list(pq, prio);
27:    }
28:    pq->pq_threads++;
29:    pthread->flags |= THR_FLAGS_IN_RUNQ;
30:
31:    PQ_CLEAR_ACTIVE(pq);
32:}
```

図 4.26: ERTF の実装

さが大きな利点となるが，第二段階では性能と安定性をカーネルの修正をしていく必要があろう．また，実装を行う OS も汎用である FreeBSD ではなく，TOPPERS 等の実時間性を考慮した実時間 OS で評価をするべきだろう．

4.4.2.2. モニタスレッドの実装

図 4.20 に示されたモニタスレッドのアルゴリズムを，図 4.27 に示すように実装した．これは仮プログラムであるが，実際に C 言語で記述されているプログラムと近い記述にしてある．

```
release_time = get_my_release_time();
pthread_set_release(pthread_self(), release_time);
do {
    current_time = get_current_time();
    if (currnet_time >= next_release_time) break;
    pthread_yield();
} while (1);
invoke_executer();
register_result_to_answer_list();
```

図 4.27: モニタスレッドの実装

4.4.3. センサデータの周期的発信に関する評価

4.4.3.1. 方針

予定されたクエリ実行時刻 (PQE_i，2 章参照) と，エグゼキュータが実行される寸前の時刻 (RQE_i，2 章参照) とのずれを測定する実験を通して，提案方式の性能を評価する．$Q_{\text{周期的発信}} = RQE_i - PQE_i$の極小化 と定式化されているから，そのずれが少ないほど性能は優れると評価される．

4.4.3.2. 準備

モニタスレッド実行する前に，KRAFT データベースにデータを挿入した．挿入したデータは INT 型とセンサ INT 型で構成され，1 タプルが 8 バイトになる．

```
% create table r (id int, s sensor_int)
% insert into r values (1, null)
```

モニタスレッドの周期は1秒，実行時間は10秒とした．すなわち各モニタ10回の実行をおこなう．これを次に示すようにしてKRAFTに登録した．

```
% create monitor release_test period 1s as select * from r
```

4.4.3.3.　環境

KRAFTのDBサーバとクライアントは別ホストに置かれる．そしてクライアントはDBサーバに対してコネクションを張り，登録されたモニタを実行させるコマンド“run monitor”を発行する．このときに実行するコマンドは，いずれのクライアントも“release_test”である．従ってエグゼキュータが取り出すデータ量は極めて少ない．また，センサデータの挿入が発生しない状況で測定を行うため，排他制御による待ち時間も発生しない．それゆえ実験環境は非常に負荷が低い環境だと考えられる．このような低負荷環境で実験する意義は，なるべく優れた評価を得るためである．そのような評価を得られれば，負荷が高くなるにつれて性能がどのように劣化するか詳細に見ることができる．

4.4.3.4.　実験結果

リリースタイムからのずれを測定し，平均をとった結果を図4.28に示す．

ラウンドロビンのリリースタイムがERTFのリリースタイムに比べて何倍悪いかを図4.29に示す．差が最小なのは並行性が200の時であり174倍である．差が最大なのは並行性が1000の時であり714倍である．

図4.28では差が大きすぎることによりERTFの結果が見えないため，図4.30にERTFの結果を示す．図4.30から，モニタ数が1000であっても，予定時刻とのずれは平均5μ秒以下であることがわかる．以上の実験結果より，ERTFはラウンドロビンに比べて圧倒的に優れた性能を示すと言える．

4.4.3.5.　議論

4.4.3.5.1.　ERTFの性能が優れる理由　提案方式がラウンドロビンよりも遥かに優れる性能をもつ理由は，ひとえに最もリリースタイムが早いスレッドが実行キューの先頭に置かれる続けるからだと考えられる．

4.4.3.5.2.　ERTFの問題点(応答の実時間性)　リリースタイムについては提案方式が優れるが，クエリ応答の実時間性については問題がある．なぜなら，結果をクライアントに送信する転送スレッドは実行キューの末尾に置かれて，全てのモニタスレッドが動作を終了するまで動くことができないからである．すなわち現在の

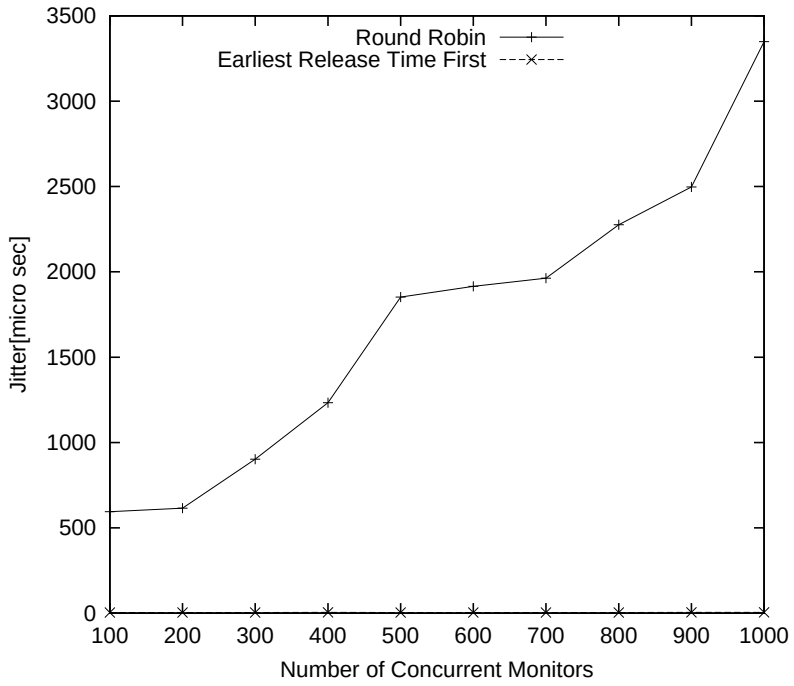


図 4.28: リリースタイムとのずれ (ERTF とラウンドロビン)

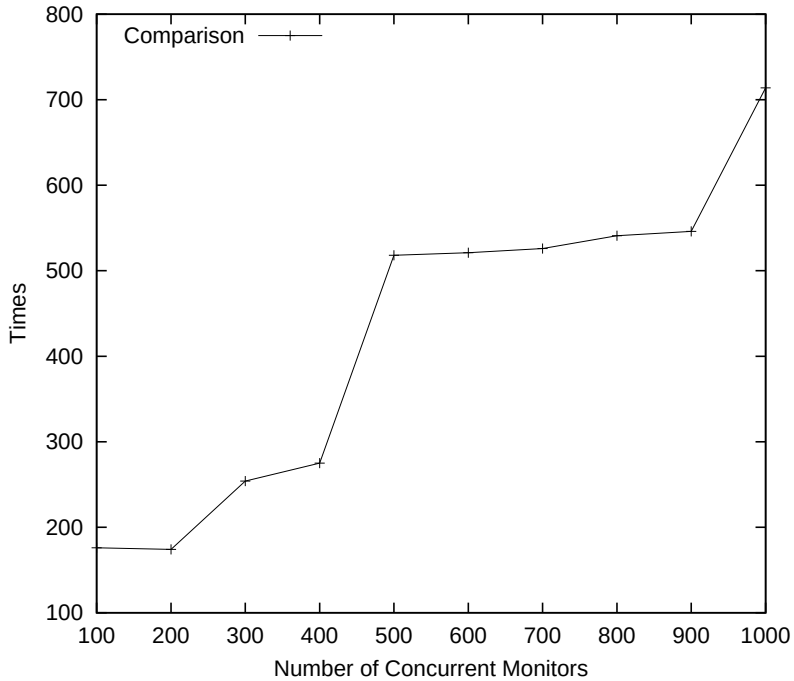


図 4.29: リリースタイムとのずれ (ERTF とラウンドロビンの比較)

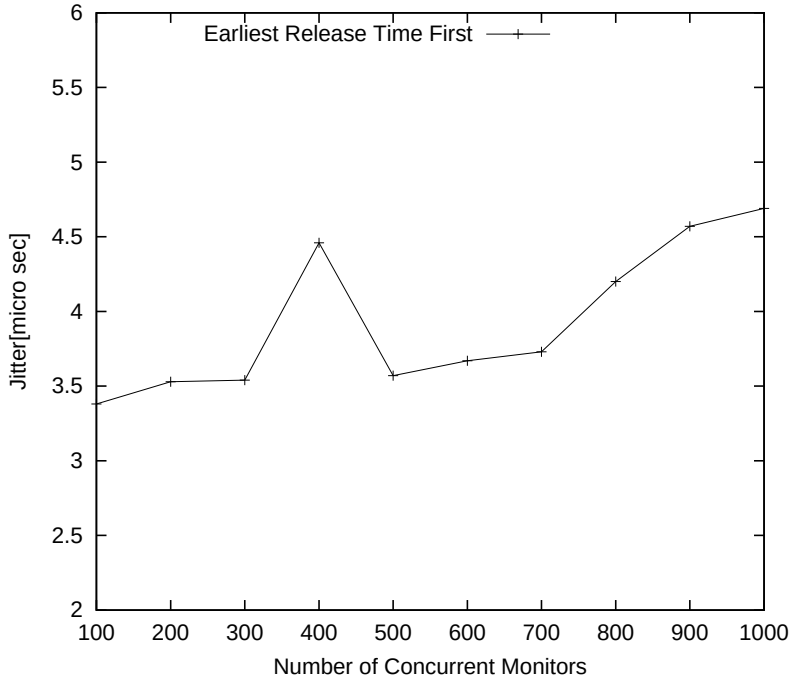


図 4.30: リリースタイムとのずれ (ERTF)

実装では，データの観測を周期的に実行することはできても，その実行結果を実時間でクライアントに提供することができない．

実行キューの先頭に置かれるモニタスレッドは，リリースタイムになるまで待機し続けるので，リリースタイムまでの時間が無駄になる．そこでその時間を転送スレッドに割り当てることで，実時間応答を実現することが考えられる．

そもそも転送スレッドが存在する理由は，モニタスレッドからクライアントへのクエリ結果転送時間を省くためであり，クエリ結果転送時間が長かった理由は通信プロトコルが重いせいであった．そこでクエリ結果をマーシャリングすることにより通信プロトコルを簡潔にし，クエリ結果転送時間を大幅に短縮した結果を図 4.31 に示す．

図 4.31 において，pipeline は転送スレッドを用いた方式を表し，sequential は逐次方式を表す．図 4.31 より，並行性が高まるにつれて逐次方式の性能はパイプライン方式の性能よりも劣化するが，並行度が 1000 であってもずれの平均は 12μ 秒以下である．これより，マーシャリングによってリリースタイムとのずれを小さくできることが示された．

しかしクエリ結果の量が増大に伴う性能劣化は避けられないだろう．その場合にもリリースタイムとのずれを減らして実時間性を保証するには，過負荷制御技法第 3.4 節を使わざるを得ない．

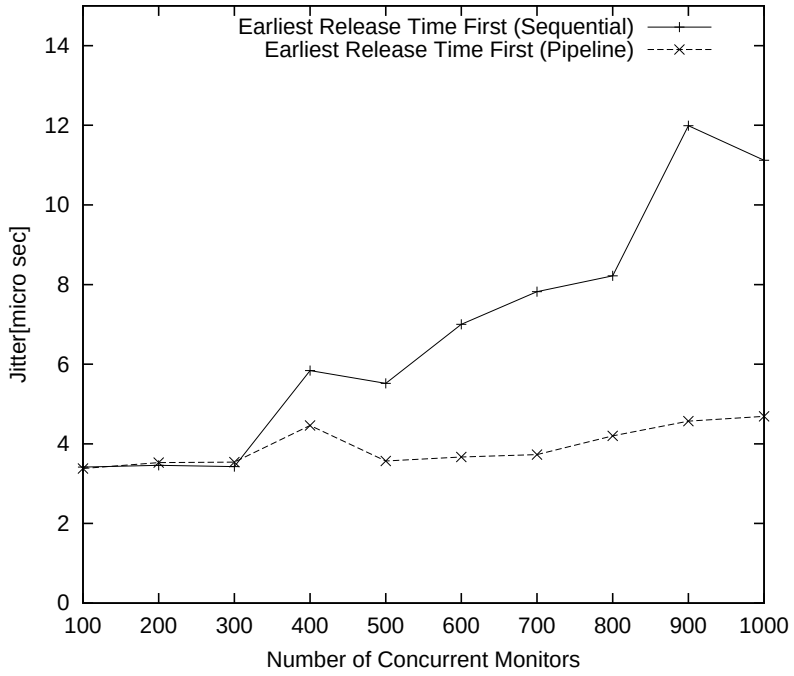


図 4.31: リリースタイムとのずれ (ERTF, パイプライン方式 対 逐次方式)

4.4.3.5.3.　センサアップロードの考慮　本節で示した実験ではセンサデータの到着が全くない環境での評価を行った．この環境は提案アルゴリズムにとっては最も良い結果を出せる環境であるが，KRAFT がセンサ応用システムに使われるときには，センサデータが到着する場合においてモニタスレッドが動作する．それゆえ，今後センサデータが到着する環境での評価をする必要がある．

4.4.4.　まとめ

本節では KRAFT における周期的発信方式 ERTF を設計および実装した．ERTF 方式はラウンドロビン方式よりも最大 714 倍優れた結果を提供し，スレッド数が 1000 であっても $(RQE_i - PQE_i)$ の平均を 5μ 秒以下に抑えた．これにより KRAFT は $Q_{\text{周期的発信}}$ を解決したと私は主張する．

4.5.　センサデータの時系列処理方式

本節では KRAFT が $Q_{\text{時系列処理}}$ を解決することを示す．

4.5.1. 設計

$Q_{\text{時系列処理}}$ を解決するために，KRAFT のセンサ型は類似シーケンス検索を許す．センサ型は $\mathsf{Dist}_{\mathsf{euclid}}(S_A, S_B)$ も $\mathsf{Dist}_{\mathsf{dtw}}(S_A, S_B)$ も許す．KRAFT は SQL のサブセットを拡張して，類似検索を行う．

例えば，“(1,2,3)” に類似した，距離尺度が $\mathsf{Dist}_{\mathsf{euclid}}(S_A, S_B) < 10$ であり，表の名前が Robovie で，属性名が sonar1 のセンサデータシーケンスを検索するには，以下のように記述すればよい．

select $*$ from Robovie where **simseq external** Robovie.sonar1 [1,2,3] **with euclid** < 10.

このクエリでは，**simseq external** は “外部シーケンスと類似したシーケンスを検索せよ” を意味する．**with euclid** は “ 距離尺度を $\mathsf{Dist}_{\mathsf{euclid}}(S_A, S_B)$ にせよ” を意味する．もしも **euclid** が **dtw** と記述されていれば，距離尺度を $\mathsf{Dist}_{\mathsf{dtw}}(S_A, S_B)$ にすることを意味する．

また，**external** は **internal** や **latest** に変えることもできる．**internal** は内部シーケンスとの比較を表し，**latest** は最新シーケンスとの比較を表す．

内部シーケンスとの比較は次のように行う．

select * from Robovie where **simseq internal**
Robovie.sonar1[1][0,0][1081238444,0] with euclid = 0

Robovie.sonar1[1][0,0][1081238444,0] の意味は，“ID が 1 で，比較元シーケンスは UNIX 時間で 0 秒 0μ 秒から，1081238444 秒 0μ 秒” である．

最新シーケンスとの比較は次のように行う．

select * from Robovie where **simseq latest 10 item of** Robovie.sonar1[1] with euclid = 0

simseq latest 10 item of の意味は，“最新の 10 個のオブジェクトから構成されるシーケンスを比較元として類似検索を行う” である．これを **simseq latest 10s of** と変更すれば，“最新 10 秒間のオブジェクトから構成されるシーケンスを比較元として類似検索を行う” になる．

4.5.1.0.4. 検索アルゴリズム　類似シーケンス検索アルゴリズムを図 4.32 に示す．このアルゴリズムは brute force 方式である．すなわちスライディングウィンドウを先頭から終端まで移動させていく間に，ユーザが指定した条件を満たす窓があれば，それを答え返却用リストに追加していく．1 行目でウインドウがセットされる．2 行目から 7 行目において検索処理が実行される．

索引が準備されていないために検索コストが高い．それゆえ今後は索引もしくは何らかの高速検索手法が実装される必要があろう．

1: Set $W(Q)$;
2: for ($i := 0$; $i < M$; i++) {
3:　　Calculate d using $W(Q)$ and $W(S_i)$;
4:　　if (d meets condition(e.g, =0, <10, *etc.*)) {
5:　　　Creates a new SENSOR_SEGMENT;
6:　　　Copies $W(S_i)$ to it;
7:　　}
8: }

Q: Query sequence of length n
S: Sequence in database constituted of M elements
S_i: Subsequence of S starting i-th element of length n
d: Distance between Q and S_i
W: Window for Q or S_i

図 4.32: サブシーケンス検索アルゴリズム

4.5.2. センサデータの時系列処理に関する評価

我々は，KRAFT への類似シーケンス検索例を 3 つ示す．

- select sonar1 from robovie
  $\rightarrow \{100, 110, 120, 130, 100, 170\}$
- select sonar1 from robovie where simseq external [100,100,100] with euclid < 10
  $\rightarrow \{100, 110, 120\}$
- select sonar1 from robovie where simseq external [100,100,100] with dtw < 10
  $\rightarrow \{100, 110, 120\}\{110, 120, 130\}\{120, 130, 100\}$

これらのサンプルクエリに関する $\mathsf{Dist}_{\mathsf{euclid}}$ と $\mathsf{Dist}_{\mathsf{dtw}}$ を表 4.4 に示す．この結果より，$\mathsf{Dist}_{\mathsf{euclid}}$ は $\mathsf{Dist}_{\mathsf{dtw}}$ よりも堅い尺度であることがわかる．

これらの実行結果より，KRAFT が $\mathsf{Dist}_{\mathsf{euclid}}$ と $\mathsf{Dist}_{\mathsf{dtw}}$ のいずれの距離尺度も許す類似検索を実行できると我々は主張する．それゆえ KRAFT は問題 2 を解決する．

なお実行時間は，アルゴリズムが brute force であるから長い．それゆえ索引や並列化による高速化が今後の課題である．

表 4.4: サンプルクエリへの $\mathsf{Dist}_{\mathsf{euclid}}$ と $\mathsf{Dist}_{\mathsf{dtw}}$

Subsequences in database	$\mathsf{Dist}_{\mathsf{euclid}}$	$\mathsf{Dist}_{\mathsf{dtw}}$
{100,110,120}	7.45	4.47
{110,120,130}	12.5	8.00
{120,130,100}	12.0	9.17
{130,100,170}	25.4	17.4

4.5.3.　まとめ

本節ではサンプルクエリを示しながら，KRAFT が $Q_{\text{時系列処理}}$ を解決することを示した．検索処理の高速化が今後の課題である．

4.6.　本章のまとめ

本章では，センサデータの高鮮度化･時系列処理･周期的発信を実現するデータベースシステムである KRAFT を提案し，KRAFT が $Q_{\text{高鮮度化}}$, $Q_{\text{時系列処理}}$, $Q_{\text{周期的発信}}$ を解決することを示した．

第5章

センサデータのインプリサイス永続化による過負荷制御

第 3.4.3 節において，センサデータの更新/追加処理に関する品質を下げることで，システムの負荷を下げる手法はまだ研究されていないことを述べた．そこで本章ではその問題に注目し，その方式によって $Q_{\text{過負荷制御}}$ を解決する手法を述べる．

5.1. 設計指針

本章では過負荷状態を制御する手法として，$\mathsf{p}(s_{unread}) \rightarrow p_{weak}$ という方針を採ることで，$\mathsf{p}(s_{read}) \rightarrow p_{strong}$ を満たしながら $\mathsf{staleness}(s_{read})$ を下げる手法を提案する．提案手法は，$\mathsf{p}(s_{read}) \rightarrow p_{strong}$ を満たすために，s_{read} のログを 2 つのリモートメモリ上のログサーバに TCP を用いて送信し，それぞれから ack を受信する．これを $\mathsf{p}_{\mathsf{tcp}}(s_{read})$ と表記する．$\mathsf{p}_{\mathsf{tcp}}(s_{read}) \rightarrow p_{strong}$ である．また，提案手法は $\mathsf{p}(s_{unread}) \rightarrow p_{weak}$ を満たすために，s_{unread} のログを 2 つのリモートメモリ上のログサーバに UDP マルチキャストを用いて一括送信し，ログサーバから ack を返させず，それゆえ ack を受信しない．このとき ack を受信しないから $\mathsf{p}(s_{unread}) \rightarrow p_{weak}$ となるが，データベースシステムが動作するマシンと，ログが送られるマシン間のネットワークを，専用プライベートネットワークにすることで，$\mathsf{p}(s_{unread})$ の成功確率を高める．さらに s_{unread} のログに抜けが発見された場合には，それを修復する．以上の処理を $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ と表記する．$\mathsf{p}_{\mathsf{udp}}(s_{unread}) \rightarrow p_{weak}$ である．

提案手法の特徴は，s_{unread} に $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ を適用することで，永続化処理の負荷を減らすことである．$\mathsf{p}_{\mathsf{udp}}(s_{unread})$ に必要な負荷は $\mathsf{p}_{\mathsf{tcp}}(s_{read})$ よりも少ないから，s の中で s_{unread} の割合が増えるほど，永続化処理の負荷を減らせる．しかも $\mathsf{p}_{\mathsf{udp}}(s_{unread}) \rightarrow p_{weak}$ であるが，ネットワークが専用プライベートであり，ログサーバとデータベースサーバの距離が 1 ホップであることから，p_{weak} は高い確率で成功する．

提案手法が s_{unread} に p_{weak} を与える理由は，将来においてアプリケーションがコミュニケーション履歴解析のために，過去のセンサデータを読み出す可能性がある

からである．ここで $\mathsf{p}_{\mathsf{udp}}(s_{unread}) \rightarrow p_{weak}$ である $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ は永続化処理を行わないことではなく，高確率で成功する永続化処理を表す．

5.2. 設計と実装

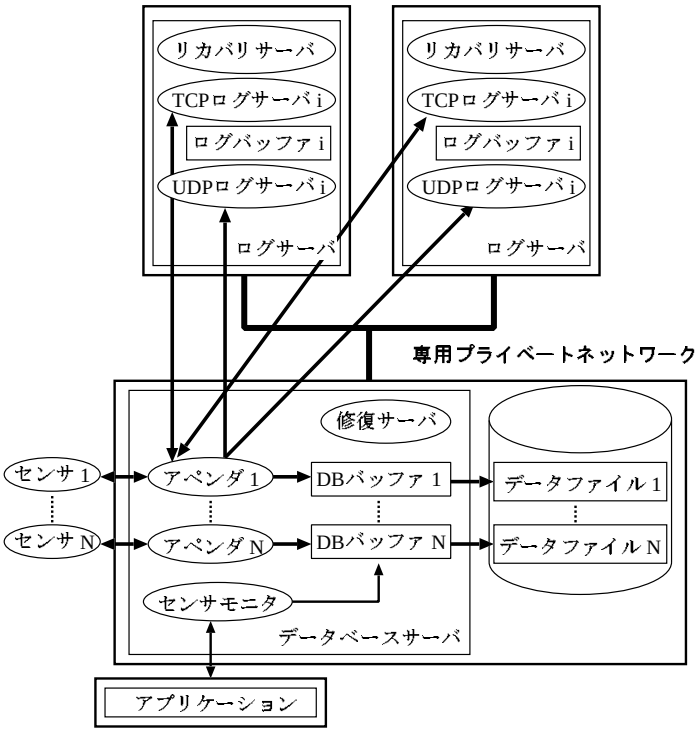


図 5.1: システム構成

本節では提案手法の設計と実装について述べる．提案手法は，図 5.1 に示されるシステム上で動作する．図 5.1 に示されるシステムは，データベースサーバとログサーバの 2 種類のサーバから構成される．データベースサーバとログサーバは 1 ホップの専用プライベートネットワークで結合される．すなわち，アプリケーションとセンサモニタをつなぐネットワークと，データベースサーバとログサーバをつなぐネットワークは異なる．

データベースサーバはアペンダスレッド，センサモニタスレッド，そして修復サーバスレッドから構成される．アペンダスレッドはセンサからセンサデータ v を取得して s を作成し，$\mathsf{p}(s)$ の後，s を DB バッファに挿入する．DB バッファ i 内の s は DB バッファ i に対応するデータファイル i に一括して書き込まれる．例えば，図 5.1 中で，アペンダ 1 は DB バッファ 1 に s を書き込み，DB バッファ 1 内の s はデータファイル 1 に書き込まれる．センサモニタスレッドは周期的に DB バッファから

表 5.1: ログパケットの構造

メンバ	説明
ストリーム ID	センサデータストリーム識別子
データバッファID	使用中のデータバッファを表現
データバッファオフセット	データバッファ中のオフセット
ログコマンド	ログサーバの実行すべき処理
s	p を与えるべきもの

最新の s を取得し，アプリケーションに提供する．修復サーバは，失われたログを修復する際に，DB バッファからデータを取得してログを作成し，そのログを TCP ログサーバへ送信する．

ログサーバは UDP ログサーバスレッド，TCP ログサーバスレッド，そしてリカバリサーバスレッドから構成される．あるアペンダスレッドに対応する，UDP ログサーバスレッドと TCP ログサーバスレッドは一つずつ存在し，両スレッドはログバッファを共有する．一方，リカバリサーバスレッドはログサーバ内に一つのみ存在する．

5.2.1. データベースサーバ

5.2.1.1. データ構造

アペンダスレッドはセンサからセンサデータ v を受信すると，$s = <at, v>$ を作成する．そして s をログパケットに変換し，ログサーバに送信する．ログパケットの構造を表 5.1 に示す．表 5.1 中のログコマンドは，ログサーバで実行すべき処理である．その処理内容には，APPEND，RECOVERY，SWITCH の 3 つがある．APPEND は s をログバッファに追加せよ，というコマンドである．RECOVERY は，リカバリ処理を実行せよ，というコマンドである．そして SWITCH は，ログバッファ内にあるバッファを切り替えよ，というコマンドである．

アペンダスレッドが s を書き込む DB バッファの構造を表 5.2 に示す．DB バッファには 6 種類，9 つのメンバが存在する．データバッファ，排他制御子，そして永続化確認バッファが 2 つずつ存在する理由は，アペンダスレッドが DB バッファ操作にダブルバッファリングを用いるからである．

5.2.1.2. アペンダスレッドのプロトコル

ここではアペンダスレッドのプロトコルについて述べる．アペンダスレッドのプロトコルを図 5.2 に示し，図 5.2 中の流れに沿って，アペンダスレッドのプロトコル

表 5.2: DB バッファの構造

メンバ	説明
ストリーム ID	センサデータストリーム識別子
データバッファID	使用中のデータバッファを表現
間引きパラメータ	プロトコルを決定するパラメータ
データバッファ 0&1	s の配列
排他制御子 0&1	データバッファの排他制御子
永続化確認バッファ 0&1	$\mathsf{p}(s)$ を実行したか示すバッファ

を説明する．

アペンダスレッドには，まず間引き率が設定される．次にバッファカウントを 0 に初期化し，データバッファ0 を施錠する (1〜3 行目)．

それからアペンダスレッドはセンサからセンサデータ v を受信し，s を作成してから $\mathsf{p}(s)$ を実行する (6 行目〜14 行目)．

7 行目では s を間引くか否かを決定する．これを決定するのは，センサモニタ起動時に設定される間引き率である．間引きが行われるのは S_{unread} のみである．例えば S_{unread} である S_i の間引き率が 50%であれば，S_i にセンサデータを追加するアペンダスレッドは，二つに一つのセンサデータに対して間引きを適用する．ここで s が間引かれるべきものならば，s は $s_{unread} \in S_{unread}$ であり，$\mathsf{p}(s) = \mathsf{p}_{\mathsf{udp}}(s_{unread})$ である．さもなければ s は $s_{read} \in S_{unread}$ であり，$\mathsf{p}(s) = \mathsf{p}_{\mathsf{tcp}}(s_{read})$ である．

その後 s をデータバッファに追加する (15〜16 行目)．このときデータバッファが一杯ならば，バッファ切り替えを行い，ディスク転送スレッドを作成する (17〜24 行目)．ディスク転送スレッドのプロトコルについては 4.1.3 節で述べる．最後に ack をセンサに送信し，センサデータ受信待ちに戻る (4 行目)．

5.2.1.3.　ディスク転送スレッドのプロトコル

図 5.2 中 23 行目において作成されるディスク転送スレッドのプロトコルを図 5.3 に示す．図 5.3 中 3 行目では排他制御子を解錠する．もしもこの解錠が終わる前に，そのデータバッファに対して，図 5.2 中 22 行目のデータバッファ施錠が行われた場合には，アペンダスレッドがブロックされる．そのような場合には，データバッファのサイズを大きくする必要がある．本研究ではこのサイズを適応的に変化させる機構を持たないため，サイズ設定はシステム設計者により委ねられる．

```
1: 間引き率を設定;
2: データバッファカウント bc を 0 に設定;
3: データバッファ0 を施錠;
4: while (1) {
5:     センサから v を受信し s を作成;
6:     ログパケット l を作成;
7:     if (s を間引くべき) {
8:         l を UDP マルチキャスト送信;
9:     }
10:    else {
11:        l を両 TCP ログサーバに送信;
12:        両 TCP ログサーバから ack を受信;
13:        bc 番目の永続化確認バッファに永続化フラグをセット;
14:    }
15:    bc 番目のデータバッファに s を挿入;
16:    bc を 1 つインクリメント;
17:    if (bc が MAX である) {
18:        両 TCP ログサーバに SWITCH を送信;
19:        両 TCP ログサーバから ack を受信;
20:        bc を 0 に設定;
21:        データバッファ変更;
22:        データバッファを施錠;
23:        ディスク転送スレッドを作成;
24:    }
25:    ack をセンサに送信;
26: }
```

図 5.2: アペンダスレッドのプロトコル

1: データバッファID が示すデータバッファをデータファイルに一括転送;
2: そのデータファイルをディスクに同期;
3: データバッファID が示す排他制御子を解錠
4: 自スレッドを消去;

図 5.3: ディスク転送スレッドのプロトコル

5.2.2. センサモニタのプロトコル

センサモニタは周期的に s_{read} の監視を実行する。このプロトコルを図 5.4 に示す．各 DB バッファ内にデータバッファは 2 つあるので，データバッファID が示さない方のデータバッファは，異なるキャッシュとして利用できる．そのため 12 行目のような処理を実行する．

センサモニタスレッドとアペンダスレッドの衝突が少ない場合には，最後に p_{strong} を与える $\mathsf{p}(s)$ が実行された s_{read} を記録しておき，センサモニタスレッドがそれを読むことが望ましい．しかし，アペンダスレッドとセンサモニタスレッドの間で衝突が頻繁に発生すれば $\mathsf{staleness}(s_{read})$ が大きくなる．また，p_{strong} を与える $\mathsf{p}(s)$ が多いほどデータバッファへの書き込み回数が増えるから，処理が重くなる．衝突が少ないアプリケーションでは，そのような手法を用いることが好ましく，衝突が多いアプリケーションでは，図 5.4 の施錠を必要としない手法を適用することが好ましい．

本論文で対象とするセンサデータストリームの到着頻度は 10ms 程度と頻繁であり，かつ本論文はセンサデータストリームの本数が複数である場合を想定するので，図 5.4 に示す手法を用いる．

5.2.3. ログサーバ

5.2.3.1. データ構造

ログサーバ内でログを保持するログバッファの構造を表 5.3 に示す．ログバッファ i は UDP ログサーバスレッド i と TCP ログサーバスレッド i の両者に共有される．データバッファと受信確認バッファが 2 つあるのは，ダブルバッファリングが使われるからである．受信確認バッファには，表 5.3 に説明があるように，データを受信したスレッドのプロトコルタイプを書き込む．データを書き込んだスレッドが TCP ログサーバスレッドならば，TCP フラグが立てられる．データを書き込んだスレッドが UDP ログサーバスレッドならば，UDP フラグが立てられる．

1: データバッファID からデータバッファをセット;
2: sbc = バッファカウント bc-1;
3: while (sbc != 0) {
4:　　if (sbc 番の s_{read} は，$p(s_{read}) \rightarrow p_{strong}$){
5:　　　sbc 番の s_read を取得;
6:　　　検索処理終了;
7:　　}
8:　　else sbc –;
9: }
10: データバッファ切り替え;
11: sbc = バッファサイズ-1;
12: 3〜9 行目の処理を実行;
13: メモリ上では見つからなかった;

図 5.4: センサモニタのプロトコル

表 5.3: ログバッファの構造

メンバ	説明
ストリーム ID	センサデータストリーム識別子
データバッファ 0&1	s の配列
受信確認バッファ 0&1	s を受信したスレッドのタイプ (TCP フラグ又は UDP フラグ)

5.2.3.2.　UDP ログサーバスレッドのプロトコル

UDP ログサーバスレッドのプロトコルを図 5.5 に示す．2 行目の s 挿入では，ログパケットのデータバッファオフセットとデータバッファID から，データバッファ内で s を挿入すべき位置を決定する．

5.2.3.3.　TCP ログサーバスレッドのプロトコル

TCP ログサーバスレッドのプロトコルを図 5.6 に示す．7 行目の s 挿入では，ログパケットのデータバッファオフセットとデータバッファID から，データバッファ内で s を挿入すべき位置を決定する．

```
1: アペンダスレッドからログを受信;
2: s をデータバッファに挿入;
3: 受信確認バッファに UDP フラグをセット;
```

図 5.5: UDP ログサーバスレッドのプロトコル

```
 1: 初期化;
 2: while (1) {
 3:     ログパケットを受信;
 4:     switch (ログタイプ) {
 5:     case APPEND:
 7:         s をデータバッファに挿入;
 8:         受信確認バッファに TCP フラグをセット;
 9:         ログ抜け確認;
10:         if (ログ抜け検出) {
11:             修復プロトコルを実行;
12:         }
13:         break;
14:     case SWITCH:
15:         データバッファ切り替え;
16:         break;
17:     }
18:     ack を送信;
19: }
```

図 5.6: TCP ログサーバスレッドのプロトコル

5.2.3.4.　修復プロトコル

ここでは，図 5.6 中の 10〜12 行における修復プロトコルについて述べる．

図 5.6 中 9 行目のログ抜け確認について述べる．受け取ったログを挿入すべきオフセットが先頭から j 番目だとする．j 番目から前方へ辿って，初めて TCP フラグがセットされている受信確認バッファのオフセットが i であるとする．$i+1$ から $j-1$ までの受信確認バッファに UDP フラグがセットされていないものがあれば，ログ抜けが検出される．

1: 修復サーバスレッドに TCP 接続;
2: 修復サーバスレッドにセンサデータ要求;
3: 修復サーバスレッドは修復データを一括受信;
4: 受信データをログバッファに挿入;
5: 接続切断;

図 5.7: 修復プロトコル

このとき，図 5.7 に示される修復プロトコルが開始される．まず，TCP ログサーバスレッドは修復サーバスレッドに TCP コネクションを張る (1 行目)．次に TCP ログサーバスレッドは，$i+1$ 番から $j-1$ 番までの範囲とデータバッファID を，修復サーバスレッドに送信する (2 行目)．修復サーバスレッドはそれを受信すると，データバッファから，要求範囲の s をまとめて TCP ログサーバスレッドに送信する．このとき TCP ログサーバスレッドが受信するパケット数は $(j-1)-(i+1)+1$ である (3 行目)．TCP ログサーバスレッドは受信パケット内の s をログバッファ内のデータバッファに挿入する (4 行目)．最後に TCP ログサーバスレッドと修復サーバスレッド間のコネクションが切断される (5 行目)．

この作業のために新たなスレッドは作成されない．なぜならばスレッド作成には時間を要するし，スレッド作成に要するコストは小さくないためにログ抜けが多発する環境では修復作業が性能を大きく劣化させてしまうし，このプロトコルはただ一度だけデータを送受信するだけの軽い処理だから，スレッドを新規に作成しない方が性能面で優れると考えられるからである．

データベースサーバ内の修復サーバスレッドは，修復時以外は何も作業も行わない．行う作業は，TCP ログサーバスレッドから TCP コネクションを張られた際の，修復作業のみである．前述の通り，修復に必要な s は DB バッファ内のデータバッファから取得するが，修復に必要な s が DB バッファ内に存在しない場合には，その s はすでにデータファイルに移行されているため，その s について $\mathsf{p}(s) \rightarrow p_{strong}$ が成立している．それゆえ，その s について修復を行う必要はないから，修復を行

わない．

提案手法では UDP ログパケットと TCP ログパケットが異なるスレッドによって受信されるために，両者の受信順序が逆転する可能性がある．逆転が起きた場合には，TCP ログサーバスレッドはログ抜けを検出する可能性があるから，データバッファ中の同領域に対して，UDP ログサーバスレッドの書き込みと TCP ログサーバスレッドの修復プロトコルによる書き込みが行われてしまう．この頻発はパフォーマンス劣化をもたらすから，受信順序の逆転は好ましくない．ネットワークの負荷が高い場合に，この現象が発生しうる．負荷が高まる問題はあるが，書き込み処理は冪等であるからデータの一貫性に関する問題は発生しない．

5.2.4.　リカバリ方式

最後にリカバリ方式を述べる．リカバリのために，データベースサーバスレッドは両リカバリサーバスレッドに対してデータファイル末尾の s を送信し，両スレッドから $\mathsf{at}(s) < \mathsf{at}(s')$ を満たす s' を取得する．そしてそれらをマージした結果をデータファイルに追加する．両スレッドから s' を取得する理由は，UDP マルチキャスト送信した s' が一方には存在し，もう片方には存在しない可能性があるからである．リカバリプロトコルを図 5.8 に示す．

1: 復旧すべき S を設定;
2: ログホスト設定;
3: S のデータファイル中の最終 s を読み出し;
4: s をリカバリサーバに送信;
5: リカバリサーバは $\mathrm{at}(s') > \mathrm{at}(s)$ である s' を送信;
6: ログホストを替えて，4〜5 を実行;
7: 受信した s' をデータファイルに追加;
8: 未処理の S があれば 1 へ;

図 5.8: リカバリプロトコル

5.3.　評価

本節では，複数のセンサが同時にデータベースサーバに v の挿入を行う環境において，周期的に s_{read} の監視を実行するセンサモニタ (図 5.1 参照) が得る $\mathsf{staleness}(s_{read})$ について，提案手法を用いた場合の結果と，比較手法を用いた場合の結果を比較する．

5.3.1. 実験条件

センサモニタはあるひとつの S_{read} のみ監視する．そして，その S_{read} については間引きが行われない．以後の記述で間引き率とある場合には，S_{unread} に対する間引き率を表す．それゆえ間引き率 100%という記述は，S_{unread} である S について，$\forall s \in S = s_{unread}$ であることを表す．センサモニタの周期は 1 秒と設定する．そしてセンサモニタの 100 回の測定をもって一実験とする．

センサデータ v は仮想センサデータ作成スレッドにより作成され，データベースサーバに TCP 経由で送られる．仮想センサデータ作成スレッドの多重度を 50, 100，150，そして 200 にした場合に，実験を行う．そして v の発生周期は 10ms に設定した．この値は Robovie に装着されているセンサの中で，最速の周期であるタッチセンサの周期と等しい．

アペンダスレッドがアクセスする DB バッファとログバッファ中のデータバッファのサイズは，いずれも 1024 に設定する．S_{unread} に対する間引きの割合は，0%，25%，50%，75%，そして 100%に設定して実験する．

5.3.2. 実験内容

次の 3 方式について $\mathsf{staleness}(s_{read})$ を測定し，その平均値，最良値，最悪値，そして標準偏差を算出した．

1. 提案方式

   提案方式は，s_{read} には $\mathsf{p_{tcp}}(s_{read})$ を実行し，s_{unread} には $\mathsf{p_{udp}}(s_{unread})$ を実行する．提案方式は，センサモニタが読むセンサ系列のセンサデータについては，全て TCP を使って 2 台の TCP ログサーバに対して永続化を行い，センサモニタが読まないセンサ系列については，間引きを行う．間引かれるセンサデータはマルチキャストにより 2 台の UDP ログサーバに送信されるが，ack は受信しない．間引かれるセンサデータは DB バッファに置かれるが，センサモニタには読ませない．

2. TCP 方式

   TCP 方式は，s_{read} には $\mathsf{p_{tcp}}(s_{read})$ を実行し，s_{unread} には $\mathsf{p}(s_{unread})$ を実行しない．すなわち TCP 方式は，s_{read} に対して，TCP を使って 2 台の TCP ログサーバに対して WAL を行う．そして s_{unread} には $\mathsf{p}(s_{unread})$ を実行しない．$\mathsf{p}(s_{unread})$ を実行されない $\mathsf{p}(s_{unread})$ は DB バッファ内のデータバッファに置かれるが，センサモニタには読ませない．

3. D-WAL 方式

   ディスクを永続化デバイスとする WAL 方式を D-WAL 方式と表記する．D-WAL 方式は，s_{read} には D-WAL を実行する．s_{unread} には $\mathsf{p}(s_{unread})$ を実行

しない．すなわち D-WAL 方式は，s_{read} に対して，D-WAL を行う．そして s_{unread} には $\mathsf{p}(s_{unread})$ を実行しない．$\mathsf{p}(s_{unread})$ を実行されない $\mathsf{p}(s_{unread})$ は DB バッファ内のデータバッファに置かれるが，センサモニタには読ませない．

5.3.3.　実験環境

ここでは，実験に用いたハードウェアおよび OS について述べる．データベースサーバと，仮想センサデータ作成スレッドは同一マシンにおいて稼働させた．そのマシンの仕様を表 5.4 に示す．ネットワークには 100M イーサネットを使用した．

表 5.4: データベースサーバと仮想センサデータ生成ホストの仕様

構成要素	説明
OS	RedHat7.3，(Kernel 2.4.18-3)
ファイルシステム	ext3
CPU	Xeon 2.4GHz シングル
ディスク	Ultra ATA 100, 7200rpm
メモリ	1GB
ネットワークインタフェース	100Mbps Fast Ethernet

ログサーバは同仕様のマシンを 2 台使用した．そのマシンの仕様を表 5.5 に示す．

表 5.5: ログサーバ用ホストの仕様

構成要素	説明
OS	RedHat7.3，(Kernel 2.4.18-3)
ファイルシステム	ext3
CPU	Pentium II 300MHz シングル
ディスク	IBM-DTTA-350640
メモリ	64MB
ネットワークインタフェース	100Mbps Fast Ethernet

データベースホストと 2 台のログサーバホストは，プライベートネットワークで結合される．このネットワークには，デーモンや arp 等のシステムレベルで送受信されるパケットを除いては，実験用パケットしか流れない．

5.3.4. 実験結果

5.3.4.1. 平均鮮度に関する結果

センサモニタが得た staleness(s_{read}) の平均値を，提案方式について図 5.9 に，TCP 方式について図 5.10 に，そして D-WAL 方式について図 5.11 に示す．図 5.9 と図 5.11 より，センサデータストリームの多重度が 200 である場合には，D-WAL 方式の間引き率 0% の staleness(s_{read}) に比べて，提案方式の間引き率 0% の staleness(s_{read}) は 61 倍程度小さく，提案方式の間引き率 100% の staleness(s_{read}) は 85 倍程度小さい．そして図 5.9 と図 5.10 から次のことがわかる．

- 提案方式は $Q_{\text{過負荷制御}}$ を解決できたこと

  図 5.9 における間引き率 100% のグラフと，図 5.10 における間引き率 0% のグラフを比較すると，センサデータストリームの多重度が上がるほど，提案方式が TCP 方式よりも優れることがわかる．この理由は，提案方式は TCP 方式よりも $\mathsf{p}(s)$ の負荷が軽いからである．間引き率 0% の TCP 方式は，全ての S 中の s に対して $\mathsf{p}_{\mathsf{tcp}}(s_{read})$ を適用するが，間引き率 100% の提案方式はある一つの S 以外の S 中の s に対して $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ を適用するため，負荷が軽い．

  しかも，本論文の実験結果では，修復スレッドが作成されなかったからログパケットの欠落はなかった．これより，センサの多重度が高い場合において，提案方式は TCP 方式よりも優れた staleness(s_{read}) を提供すると同時に，等しい $\mathsf{p}(s)$ を提供できた．センサデータストリームの多重度が 200 である場合に，提案方式の間引き率 100% は TCP 方式の間引き率 0% に比べて 32% 程度優れた staleness(s_{read}) を提供できた．以上より，提案方式は本論文で解くべき問題である「$\mathsf{p}(s_{read}) = p_{strong}$ を満たしながら staleness(s_{read}) を下げること」を解決できたことがわかる．

- staleness(s_{read}) の逆転現象

  図 5.9 を見ると，グラフの変化が単純ではないことがわかる．図 5.9 において，間引き率が 75% と 100% の場合において，多重度が 50 から 100 に上がっているにも関わらず，staleness(s_{read}) が下がっている．また，間引き率が 75% のグラフについて，多重度が 50 と 150 の場合の staleness(s_{read}) を比較すると，150 の方が低くなっている．

  この逆転現象が起きた理由は，システムの負荷が限界点に達していないことだと考察される．データを細かく見ると，間引き率が 100% であり，多重度が 50 の場合に記録された最悪の staleness(s_{read}) は 111.739 ミリ秒だった．この値は平均値である 14.9 ミリ秒の 7 倍以上大きい．最も負荷が低い実験条件においても，このような特異な実験値が発生した．一方，多重度が 200 である場合には，間引き率が 0% と 25% の場合と，間引き率が 100% の場合につい

て，staleness(s_{read}) の平均値が明確に異なる．この理由は，間引き率が 0% と 25% で多重度が 200 の場合には，負荷が限界点に達していることだと考察される．

以上より，本論文での実験条件においては，負荷が限界点に達していなければ，特異値が発生することにより，staleness(s_{read}) の平均値は安定した傾向を示さないのだと考察する．特異値が発生することは，5.4.2 節で示される staleness(s_{read}) の標準偏差値と最悪値に大きな幅があることからも理解できる．特異値が発生する理由は，アペンダがデータを持っているにも関わらず，カーネルのスレッドスケジューリングのために，それを DB バッファへ書くのが遅れ，それによりセンサモニタが新しいデータを読めなかった可能性が考えられるが，詳細な解析は今後の課題である．

図 5.10 においても図 5.9 と同様に，グラフの変化が単純でない結果が見られる理由は，図 5.9 同様に，負荷が限界点に達していないことだと考察する．

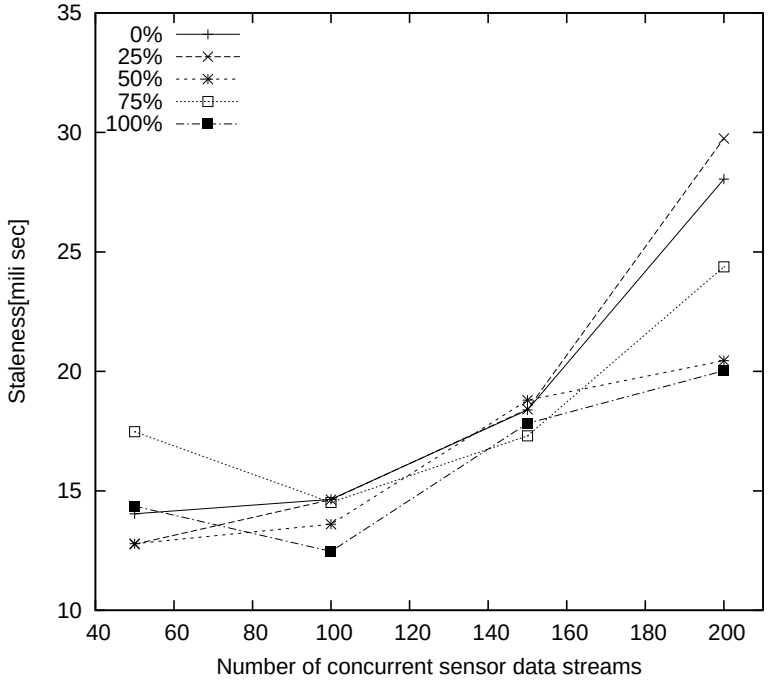


図 5.9: 提案方式 (staleness(s_{read}) の平均)

5.3.4.2.　最良値，最悪値，および標準偏差値

鮮度に関する最良値，最悪値，そして標準偏差値について，提案方式，TCP 方式，そして D-WAL 方式のそれぞれについて，実験で得られた結果を表 5.6 に示す．

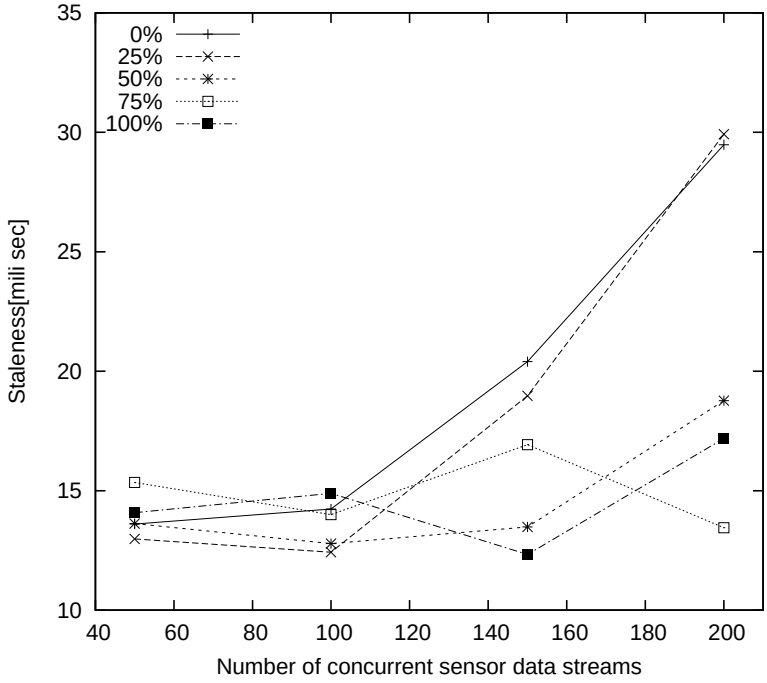


図 5.10: TCP 方式 (staleness(s_{read}) の平均)

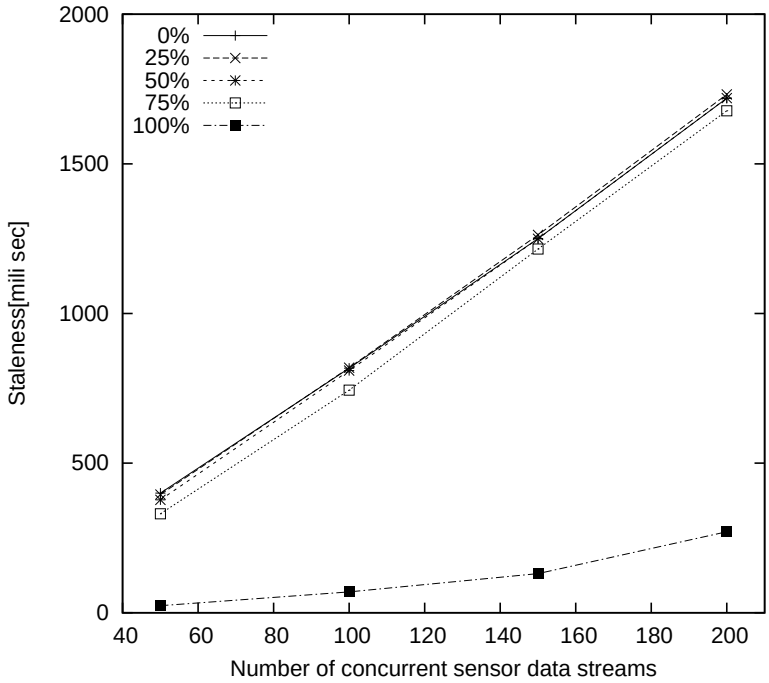


図 5.11: D-WAL 方式 (staleness(s_{read}) の平均)

表 5.6: 鮮度に関する最良値，最悪値，標準偏差値の範囲

	プロトコル	範囲 (msec)
最良値	提案方式	0.862〜6.760
	TCP 方式	1.112〜7.224
	D-WAL 方式	6.138〜1160.122
最悪値	提案方式	28.236〜413.408
	TCP 方式	30.035〜652.908
	D-WAL 方式	547.555〜4231.027
標準偏差値	提案方式	5.804〜40.709
	TCP 方式	5.752〜64.306
	D-WAL 方式	149.046〜7051.912

表 5.6 より，リモートメモリを用いた方式である提案方式と TCP 方式は最良値，最悪値，そして標準偏差値のいずれについても実験で得られた値の範囲が同程度であり，その幅が 22 倍程度に納まっていることがわかる．一方，D-WAL は最良値の幅が 190 倍程度，最悪値の幅が 7 倍程度，そして標準偏差値の幅が 47 倍程度あることがわかる．

5.4. 議論

5.4.1. パケットが落ちる確率

アペンダスレッドは $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ についてはログサーバスレッドからの ack を受け取らず，修復作業をおこなう．修復作業の間は TCP ログサーバはブロックされるが，修復作業が行われる可能性は低い．本論文における実験の限りにおいては，修復作業は一度も実行されなかった．この理由について考察する．

1 パケットが x 台の端末全てに誤りなく転送される確率は，ビットエラー率を e とすると，$(1-e)^x$ となる [宮本 他 01]．有線環境でのビットエラー率は 10^{-9} を下回る [中井 他 00]．ログパケットのサイズは，ログパケット自体が 48 バイトであり，それに付加される UDP ヘッダ，IP ヘッダ，イーサヘッダ，そして CRC のサイズがそれぞれ 8，20，14，4 バイトだから，ログパケットのサイズは合計で 736 ビットとなる．

このとき，1 パケットが x 端末全てに届く確率 $Probability(x)$ は，次のように計

算される [宮本 他 01].

$$\begin{aligned} Probability(x) &= (1-e)^x \\ &= (1-736\times10^{-9})^x \end{aligned}$$

本実験においてリモートメモリ数は 2 であったから，$x=2$ として上式に代入すると $Probability(2)$ が次のように得られる．

$$\begin{aligned} Probability(2) &= (1-e)^2 \\ &= (1-736\times10^{-9})^2 \\ &\simeq 1-1.5\times10^{-6} \\ \therefore 1-Probability(2) &\simeq \frac{15}{10^7} \end{aligned}$$

これより UDP パケットが落ちる確率は $\frac{15}{10^7}$ 程度となるので，666667 個にひとつのパケットは落ちることになる．実験ではセンサデータの発生周期が 10msec であったから，1 秒間に 100 個のセンサデータが発生するので，約 6667 秒，1 時間 51 分に一度はパケットが落ちる計算になる．従って，このとき間引き率が 100% である S_{unread} が 200 本あれば，6 分に 1 つはパケットが喪失したはずであるが，本実験ではパケット喪失が発生しなかった．

5.4.2. 提案手法の限界点

本論文における実験では，高負荷時において，提案手法は比較手法よりも小さな staleness(s_{read}) を提供することが示されたが，その限界性能は明らかにならなかった．そこで，ここでは提案手法のパフォーマンスが限界となりうる点について議論する．

リモートメモリを永続化デバイスとしているから，ボトルネックになるのはアペンダスレッドとログサーバスレッド間のトラフィック量である．このトラフィックが増大した場合に，性能が限界に達する可能性がある．そして，限界の達し方には二通りがある．一つはログホストのスレッド処理能力であり，もう一つはネットワーク帯域幅である．

スレッド処理能力が十分であり，ネットワーク帯域幅が不足である場合には，パケットが落ちる可能性が高まる．それゆえ TCP ログパケットについては再送がなされるから，ネットワークトラフィックがさらに発生する．同時に UDP ログパケットが失われる可能性も高まるから，修復作業が頻繁に行われ，これがネットワークトラフィックをさらに発生させる．この悪循環に陥れば，staleness(s_{read}) は時間の経過につれて大きくなる．

逆に，スレッド処理能力が不足であり，ネットワーク帯域幅が十分である場合には，ログホスト上で仮想メモリが使われたり，スレッドスイッチの頻発により，

CPU 時間が各スレッドに十分与えられず，ログホスト上での処理時間が長くなる．それゆえ，$\mathsf{p}(s)$ に要する時間が長くなるから，$\mathsf{staleness}(s_{read})$ は大きくなる．

いずれの場合においても $\mathsf{staleness}(s_{read})$ は劣化するから，今後の研究において，限界点における負荷制御手法の導入が望まれる．

5.5. 本章のまとめ

本章では，複数台のリモートホストのメモリに対して WAL を適用し，さらにアプリケーションに使用されていないセンサデータストリームの一部に対して，投機的な WAL を適用し，処理負荷を減らす手法を提案した．そして提案手法を実装し，提案手法と TCP 方式について比較実験を行った．その結果，センサデータストリームの多重度が高い場合に，提案手法は TCP 方式より最大 32%優れた鮮度を提供し，さらに等しい永続性をもつデータを提供できた．この結果より，提案手法は $Q_{\text{過負荷制御}}$ を解決できたと私は主張する．そして実験結果においてパケット欠落が生じなかった理由と，提案手法のボトルネックとなりうる点について議論した．

この手法はセンサデータの更新/追加処理を品質を下げながら実行することによりシステムの負荷を下げる手法である．そして第 3.4.3 節において述べたように，私の知る限り初めて研究された手法である．

第6章

議論

本章ではこれまでの論文内容に関して議論する．まず，そもそもどうしてセンサデータを即時永続化し，遅延評価をしないのか議論する．次に，本論文の課題について議論する．それから応用について議論し，最後にデータベースシステム開発論について議論する．

6.1. センサデータ即時永続化の是非

6.1.1. 即時永続化と遅延評価の比較

本論文ではデータベースシステムに到着したセンサデータを高鮮度化するために永続化処理の高速化手法を提案したが，この方式は図 6.1 のように図示できる．図 6.1 では全てのセンサデータはアプリケーションに届く前に永続化される．

一方，センサデータを高鮮度化するには遅延評価法が考えられる (図 6.2)．これは永続化処理をせずにアプリケーションにセンサデータを与えることでセンサデータを高鮮度化し，ある程度センサデータがバッファにたまったら，それらを一括して永続化する方式である．

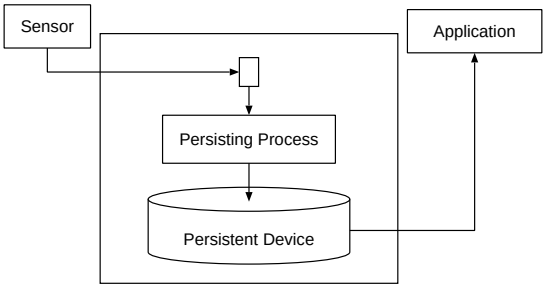


図 6.1: 非遅延評価方式 (KRAFT モデル)

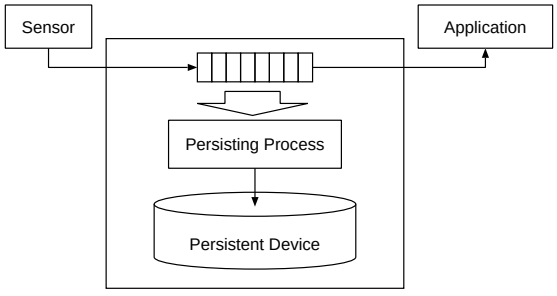


図 6.2: 遅延評価方式

KRAFT 方式よりも遅延評価方式の方が優れた鮮度をセンサデータに提供できると考えられるにも関わらず KRAFT は遅延評価方式を使わない理由は，遅延評価

方式ではアプリケーションにより読まれたセンサデータをデータベースシステムが失う可能性があるからである．このような事態は，アプリケーションに提供されたが永続化されずにバッファリングされているセンサデータが存在する時に，データベースシステムが故障した場合に発生する．このときバッファリングされていたデータをデータベースシステムは全て失う．

このような事態が発生すると次のような問題が発生する．

1. データベースシステムを経由したデータが消失すること

   データベースへのアクセスはトランザクションとして処理される．このトランザクションの性質には原子性，整合性，分離性，そして耐久性があり，あるデータベースシステムの教科書 [Date 97] では耐久性について次のような記述がある．「耐久性 (durability)：トランザクションがコミットされた場合，たとえその直後にシステムがクラッシュしても，その変更は生き残る」．それゆえデータベースシステムから得たデータを再度検索したときにそれは既に失われていることは，トランザクションの観点から許されない．

   データベースシステムを経由することでデータ鮮度が劣化して使いものにならない場合には図 6.3 に示すようにセンサデータをアプリケーション X が読んだあとに，複数のデータをデータベースシステムに一括書き込みすればよいだろう．ただしこの場合には次のような問題が生じる．(1)X 以外のアプリケーションには低鮮度センサデータが与えられること．(2)X はデータベースシステムへの書き込み処理を実行する手間が必要になること．

2. データ解析精度の劣化

   センサデータが永続化される理由の一つには，蓄えられたセンサデータを後で解析したい要求があることだった．遅延評価モデルを用いるとバッファ内のデータは障害発生時に失われてしまうため，即時永続化モデルに比べて解析の際に精度が劣化する可能性がある．

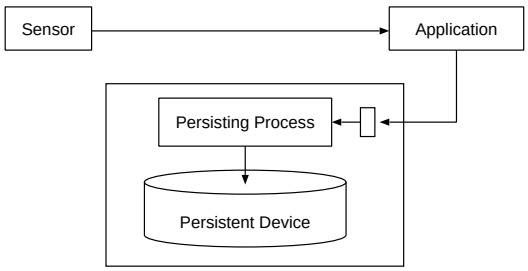


図 6.3: データベースシステムへの遅延書き込み

以上の議論より，データベースシステムとしてセンサを扱うならば遅延評価モデルではなく即時永続化モデルにせざるを得ないと考えられる．

6.1.2. インプリサイス永続化処理と遅延評価の関連性

第 5 章ではインプリサイス永続化処理について述べたが，同手法は遅延評価の一種だと考えることができる．インプリサイス永続化処理のモデルを図 6.4 に示す．図 6.4 ではバッファ内に白いオブジェクトと黒いオブジェクトがある．白いオブジェクトは即時永続化され，アプリケーションから見ることができる．一方，黒いオブジェクトは即時永続化されず，後で一括して永続的デバイスに書き込まれるので，バッファにある間はアプリケーションからは見ることができない．ただし永続化デバイスに書き込まれた後はそこから読み出すことができる．

図 6.4 と図 6.5 の関連性を考えると，図 6.5 は図 6.4 における黒いオブジェクトを読める方式であると考えることができる．つまり図 6.4 に示されるインプリサイス永続化処理は**永続化されないデータをクライアントに読ませないことにより，耐故障性に関するトランザクションとしての問題**[1]**を回避している**とも表現できる．

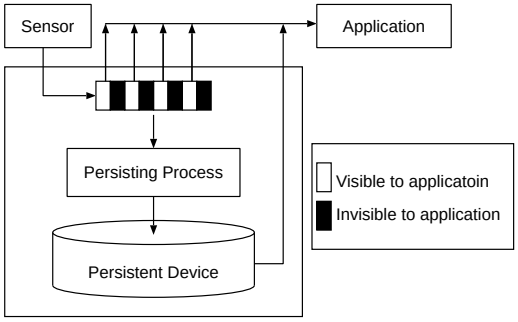


図 6.4: インプリサイス永続化処理

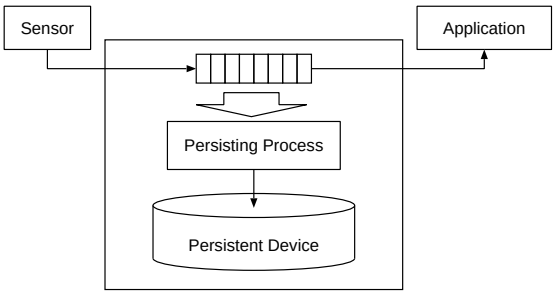


図 6.5: 遅延評価方式 (再掲)

また，図 6.4 に示したインプリサイス永続化処理をオペレーティングシステムのファイルバッファと比較して見直すと，それはライトスルー法と変形ライトバック法を組み合わせた手法だとも考えられる [Silbershatz *et al.* 02]．図 6.4 における黒いオブジェクトは即時永続化されるためにライトスルー処理されると考えられる．一方，同図における白いオブジェクトは即時永続化されずにダーティビットを振られ，一括して永続的デバイスへ書き込まれるためにライトバック処理される．これが通常のライトバックと異なる点は，通常のライトバックでは閲覧可能なオブジェクトが，インプリサイス永続化処理手法におけるライトバック処理では閲覧不可能な点である．オペレーティングシステムにおけるファイルバッファとデータベースシステムの性質が異なる理由は，アクセス方法がトランザクションであるか否かに依ると考えられる．

[1]第 6.1.1 節参照.

6.2.　今後の課題

6.2.1.　高鮮度化に関する課題

センサデータの高鮮度化に関する本研究の貢献は，リモートロギングをデータベースシステム KRAFT の機能として実現し，実現方式が優れた鮮度を提供できることを実証実験を通して示した点にある．

本研究の課題は，KRAFT では 1 台のリモートホストしか使用しないために，ログレコードの永続性を強化する必要がある点である．

永続デバイスの永続性を強化する方法には (1) 複数台のリモートホストを使用する方式と，(2) 不揮発メモリを用いる方式の 2 つが考えられる．以下ではこれらの方法について議論する．

6.2.1.1.　永続デバイスとして複数台のリモートメモリを使用する手法

永続デバイスとして複数台のリモートメモリを使用する手法が考えられる．第 4 章では 1 台のリモートホストを用いた主記憶 WAL を KRAFT において実現し，データ鮮度および追加処理時間について評価を行った．

複数台のリモートホストを用いた主記憶 WAL の実現方式はすでに ClustRa に関する文献 [Hvasshovd *et al.* 95] で述べられているが，同文献ではひとつの方式が提案されたに留まっており様々な手法についての議論がされていない．それゆえデータ鮮度，応答時間，スループット，そして耐故障性の観点から主記憶 WAL は様々に研究される必要があると私は考えている．

6.2.1.2.　永続デバイスとして不揮発メモリを使用する手法

永続デバイスとして電源障害時にも記録を保持できる不揮発メモリを使用する手法が考えられる．

現在，MRAM や FeRAM などの不揮発メモリの実用化がなされてきている．これらの不揮発メモリは，通常の主記憶として利用されるメモリと同様のアクセス方法でアクセス可能であり，かつ予備電源を用いることなく不揮発性を保持する．

これらの不揮発メモリは，SRAM と同様の方法でのアクセスが可能であり，容量や速度，書き換え可能回数などの面からも主記憶として十分に利用可能なものである．表 6.1 に主な半導体メモリの性能比較を示す．表 6.1 からもわかるように，不揮発メモリの中で MRAM はとりわけその速度 (読み出し，書き換え時間) に関して，現行の計算機システムで主記憶として用いられている DRAM の性能と比べて遜色がない．また，書き換え可能回数や消費電力，さらには集積率も遜色のないものであると言える．以上より，将来のメモリは全て MRAM や FeRAM などの不揮発メモリによって置き換えられる，との予測も存在する．よって，将来計算機シ

ステムの主記憶が，これらの不揮発メモリで構成されることは十分に予測されることである．

表 6.1: メモリデバイスの性能比較 (「情報処理」Vol.45 No.1 pp.43 より)

	MRAM		FeRAM	NAND Flash	DRAM
	Type A*	TypeB†			
不揮発性	○		○	○	×
書き込み速度 (ns)	10		20〜30	200,000 (per page)	50
読み出し速度 (ns)	500	50	20〜30	Random Access 4,000	50
セルサイズ (DRAM との相対値)	0.6	1.3	2	0.6	1
書き換え耐性	No-limitation		10^{12}	10^{6}	No-limitation
消費電力 (mW)	100〜400		〜10	100	400
動作電圧 (V)	1 以下		2.5/1.2	12	2.5

*クロスポイント型
†選択トランジスタ型

そのような場合には，データベースシステムはリモートメモリではなく不揮発メモリを永続デバイスとして利用することになることも同時に予測される．

6.2.2. 周期的発信に関する課題

本論文の周期的発信に関する貢献は，KSE を用いて周期的発信を実現することにより，モニタ数が 1000 の場合でも平均ギャップを 12μ 秒程度に抑えられることをセンサデータベースシステム KRAFT において実証できた点である．

スレッドスケジューリングの実現手法は 2 つあり，それらはユーザレベルでの実現とカーネルレベルでの実現である．スレッドスケジューリングをカーネルレベルで実現すれば優れた性能を出せる利点があるものの，カーネルコンパイルに長時間を要すると共にバグが引き起こす OS 停止により開発期間が長期化するという欠点がある．一方，スレッドスケジューリングをユーザレベルで実現すればコンパイル時間をカーネルレベルでの実現に比べて短縮できると同時にバグでも OS が停止するおそれはないという利点があるものの，カーネルレベルでの実現に比べて低い性能しか出せないという欠点がある．

本論文ではユーザレベルでの実時間スケジューラを実現し，実現方式により $Q_{\text{周期的応答}}$ を解決することを示したが，より優れた性能を出すためには，カーネルレベルでのスケジューリングに手を入れたり，実時間 OS を用いる必要があるだろう．

6.2.3. 時系列処理に関する課題

6.2.3.1. 距離尺度

本論文では状況認識を支援するために KRAFT に類似シーケンス検索手段を提供させた．このときに距離尺度として $Dist_{euclid}$，$Dist_{dtw}$ を提供した．

しかし KRAFT は SoundCompass[小杉 他 04] で用いられている City-block 距離 [高根 80] および HURMA[本田 03] で用いられている LCSS 距離を提供しない．それゆえ今後は両距離尺度を提供する必要がある．

6.2.3.2. 時系列データへのアノテーション

SoundCompass や HURMA などのセンサ応用システムには共通点がある．それは原データに何らかのフィルタ処理をかけて特徴抽出を行い，その特徴をキーとして原データを検索することである．これを図 6.6 に示す．図 6.6 では原データから特徴抽出をおこなうことにより時系列メタデータが得られ，時系列メタデータから原データへのポインタが張られている構成を示している．

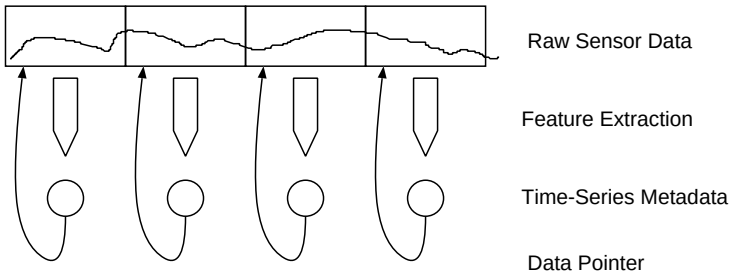


図 6.6: 時系列アノテーション

この処理を支援するには，データベースシステムは様々なフィルタ処理手法を支援すると共に，抽出された特徴をアノテーションとして原データへ貼り付けられる機能を提供する必要があるだろう．

この機能はセンサ応用システムを支援するために重要であると考えられる．

6.2.4. 過負荷制御に関する課題

過負荷制御に関する本研究の貢献は，従来は考えられてこなかったインプリサイス永続化方式の一手法を提案し，専用実験システムを通してその有効性を示したことにある．

第 3.4.3 節で示したように，過負荷制御技術には 4 種類ある．それぞれ一長一短であり，どれか一つが優れているというわけではない．それゆえ場合に応じてこれらの手法を組み合わせることが望ましい．

過負荷制御が困難である理由は，過負荷制御方式はアプリケーションによって変わり，さらにアプリケーションの状況に応じても変わることである．それゆえデータベースシステムは全ての過負荷制御技術を持っておき，アプリケーションも過負荷制御時に許す品質劣化 (例えば実時間性や鮮度) をデータベースシステムに知らせておき，過負荷制御時にはデータベースシステムがアプリケーションの許す範囲で品質劣化を行うような過負荷制御を選択する技法を考えることが望まれる．

6.2.5.　KRAFT の総合的

上では各機能毎に課題を述べてきたが，総合的に KRAFT を見直した時には次の課題がある．

6.2.5.1.　並列化

本研究では単一 CPU 上で動くセンサデータベースシステムのプロトタイプとして KRAFT を開発したが，地震データのように大量に生じるデータに対応するためには並列化をおこない性能を向上させる必要があると考えられる．

6.2.5.2.　組込化

一方，移動機や PDA に KRAFT を組み込む場合には，データ溢れ時の対応や省電力化およびサーバとの効率的連携機能の実現が要求されると考えられる．

6.3.　応用

6.3.1.　KRAFT の適用可能性と有用性

本論文で私は KRAFT がコミュニケーションロボット Robovie に対して適用可能であり，かつそれが有用であることを示したが，KRAFT の適用範囲は Robovie に留まらず，センサデータの高鮮度化・周期的発信・時系列処理をデータベースシステムに求めるアプリケーション全般に適用できると考えられる．例えば，ロボット-センサネットワークおよび実時間地震同定システムはデータベースシステムに対して 3 機能を要求するため，KRAFT は適用可能である．ただし有用性は完全でない．これを以下で述べる．

6.3.2.　適用例

6.3.2.1.　地震データ監視システムへの適用

地震データ監視システムでは，一日に数 GB 程度得られる地震データを実時間に解析して，地震伝播パタンを予測する．このシステムに KRAFT を適用するために

私は地震研究機関と共同研究を始めたところである．

得られたばかりのセンサデータはオンライン解析に使われるため鮮度が求められる．さらにそのデータはデータベースに蓄えられて将来の解析時に検索される可能性があるため，永続化される必要がある．また監視は周期的に実行されることが求められるため周期的監視機能が要求される．予測アルゴリズムのひとつとして過去の類似地震波の検索という手法があるため，類似シーケンス検索機能が必要になるだろう．

ただし，予測アルゴリズムは類似地震波検索以外にも自己回帰モデルへのあてはめなどの他の手法もあるため，類似シーケンス検索機能以外にも様々な手法を提供する必要があると考えられる．

6.3.2.2.　センサネットワークへの適用

私は現在 MOTE[XBOW 04] を用いてセンサネットワークを構築している．この目的は人間がインタラクションしやすい環境の開発である．

システム開発者は時々刻々と変化するセンサネットワークの状況をすぐに見たいため，データ鮮度と周期的監視機構が要求される．そして，オンラインまたはオフラインにおいてセンサデータの類似性から類似したインタラクションを同定し，それをもとに一連のインタラクションのメカニズムを解析するため，データの永続性と類似シーケンス検索機能が要求される．

ただし，現在の KRAFT は画像センサ型をサポートしないため，カメラから得られる画像センサデータを扱うことができない．

6.3.2.3.　適用可能性と有用性

上記 2 つのシステムには，データ鮮度，データ永続性，周期的監視機能，類似シーケンス検索が要求される．それゆえ，機能 (1)：データの永続性と優れたデータ鮮度，機能 (2)：類似シーケンス検索機能，機能 (3)：周期的監視機能が求められるため，実時間地震同定システムとロボット-センサネットワークに KRAFT は適用可能であり，ある程度有用だとも言える．

しかし有用性は完全ではない．そこで有用性を増すために，KRAFT は実時間地震同定システムが求める類似シーケンス演算以外の解析手段を提供し，ロボット-センサネットワークにおいてカメラを支援するための画像型を提供する必要がある．

6.4.　データベースシステムがセンサを扱うには

6.4.1.　データベースシステムは如何に発展すべきか

センサデータベースシステムの目的はセンサ応用システムの支援である．それゆえそれらのシステムが求める機能を実現する必要がある．これを実現するには様々なセンサ応用システムがどのような機能を提供するか，そしてどのように構成されているのかを綿密に調査し，そして必要な機能を抽出してデータベースシステムにおいて効率的に実現することが必要である．以下ではデータモデルおよび提供すべき機能について述べる．

6.4.1.1.　データモデル

いままでは関係データモデル，オブジェクト指向データモデル，そしてオブジェクト関係データモデルが考えられてきた．関係データモデルの長所は集合論に基づくためにデータ型と演算が単純なことであり，欠点はデータ型としてテーブルしか扱えず，それらに対してほとんど関係演算しか実行できないことである．オブジェクト指向データモデルの長所はプログラミング言語程度の自由度があるために幅広いデータ型や演算を定義できることであり，欠点はそのために基本的なデータ構造を決めることが困難であったり直観的に分かりやすいインタフェースを作成することが困難なことである．それゆえ私は関係データモデルからもオブジェクト指向データモデルからも歩み寄ったモデルであるオブジェクト関係モデルがセンサデータベースシステムには適切だと考える．これを図 6.7 に示す．

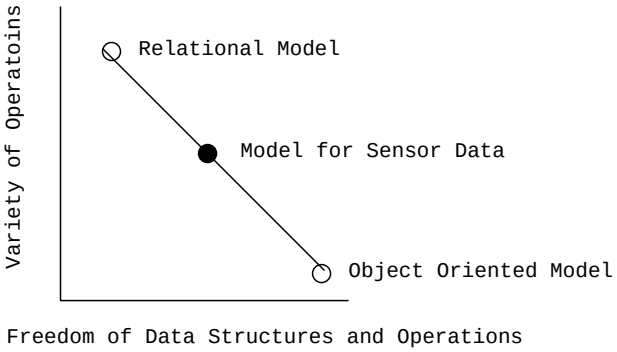


図 6.7: センサデータベースシステム向けのモデル

さて，このような方針をとると，関係 DBMS に比べてセンサデータベースシステムの機能は増大し，データベースシステムは巨大化するだろう．なぜならオブジェクト関係モデルはオブジェクト特有の演算を与えられるために，膨大量の演算ライブラリを提供する必要があるからである．しかもセンサ応用システムの種類が

増加するに伴い，センサデータ用オブジェクトが提供すべき演算数は増大せざるを得ない．このような方向を推進するには，データベースシステム開発者に多大な労力が課せられることになるが，それによりセンサ応用システムの開発工程を減らせるのだから，その方向への推進は正しく好ましいものであると私は考える．

6.4.1.2.　提供すべき機能

データベースシステムはあらゆる情報システムの中核となる基盤システムであるため，データを管理するだけではなく，データを解析処理することも求められる．従ってデータベースシステムのターゲットに応じて異なる解析機能をデータベースシステムは提供するべきだと考えられる．本論文ではセンサ応用システムがデータベースシステムのターゲットであるため，KRAFT はセンサ応用システムが要求する解析処理機能である類似シーケンス検索処理機能を提供した．これからもターゲットが必要な解析処理が増えるについてデータベースシステムが提供すべき機能は増加し続けるだろう．

機能から見たターゲットとデータベースシステムの関係を図 6.8 に示す．この関係が変わることはないだろうと私は思う．なぜならばデータベースシステムはデータを蓄えるだけではなく，その解析，表現，そして検索に関わるからである．その意味でデータベースシステムはファイルシステムと異なる．ファイルシステムはデータの内容には全く踏み込まない．

データベースシステム以外の基盤システムにはコンパイラ，ネットワーク，そしてオペレーティングシステムがあるが，これらがデータベースシステムと異なるのは，データを内容に考慮した処理をしない点である．データベースシステムはデータの内容に応じた解析処理を支援するからこそ，常に新しい機能拡張が要求されると私は考えている．

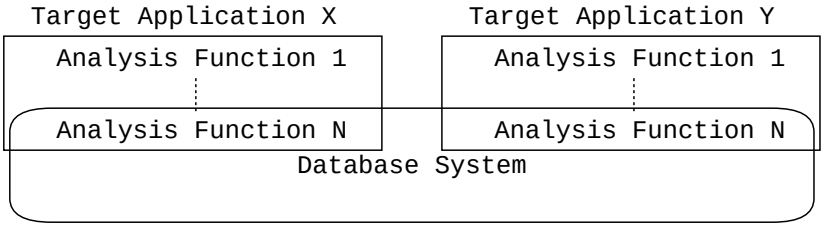


図 6.8: 機能から見たターゲットと DBMS の関係

6.4.2.　データベースシステム開発者が求める技術

データベースシステムが強大になるには，豊富な機能を効率的に実現しなければならない．それにはデータベースシステムを開発するツールが重要な役割を果たすと

考えられる．私はコンパイラが重要なツールだと考えている．

近年，性能向上を追求するためにCPU キャッシュを意識したデータ管理 [Shao *et al.* 04] が考えられている．その中にはプリフェッチ命令を利用して CPU キャッシュを有効利用する研究 [Chen *et al.* 04] もある．こういった研究ではアセンブラを使って命令を記述せざるを得ないが，アセンブラでデータベースシステムを開発することは莫大な労力を要する．そこで高次言語で記述されたシステムを高性能に翻訳してくれるコンパイラがデータベースシステム開発者には望まれる．ただしこういったコンパイラの開発は極めて困難であろう．

前述したコンパイラは C 言語や Java 言語のコンパイラを想定していたが，さらに高次の仕様レベルから記述ができ，それらを高性能なバイナリに変換してくれるコンパイラがあれば，開発労力が大きく削減されて多様なデータベースシステムを手軽に作れるようになるだろう．その結果として研究が進み，よりよいデータベースシステムが開発されるようになるのではないだろうか．

データベースシステムは開発が困難なソフトウェアである．なぜならそれはコンパイラ，オペレーティングシステム，そしてネットワークという様々な要素を持っているからである．今まではそれ程多くのデータベースシステムの開発がされてこなかったと考えられるが，第 1.1 節で述べたようにデータベースシステムは社会で幅広く使われているソフトウェアなのだから，優れた開発ツールを提供することにより開発コストが低下されることは，社会的貢献という観点から大切であると考える．

6.4.3.　センサデータベースシステムに関する誤解

ここではセンサデータベースシステムに関する誤解について述べたい．

1. センサデータベースシステムの提供する機能はすでに各アプリケーションで開発されているのだから，センサデータベースシステムは不要である

   基盤ソフトウェアの存在意義は，アプリケーション開発を効率化することにあるのだから，各アプリケーションで開発されてきた機能をセンサデータベースシステムが持つのは必然である．これは第 3.3.2 節においても述べた．

   例えば，仮想メモリは昔 OS に提供されなかったために各アプリケーションプログラマはスワップモジュールを開発をしていたが，現在仮想メモリは OS に提供されるためにアプリケーションプログラマはスワップモジュールを考えることなくアプリケーションを開発できる．

2. センサネットワークはセンサデータベースシステムである

   小型センサを張りめぐらして構築したセンサネットワークにおけるデータ処理手法が近年数多く研究されている [Madden *et al.* 03, Tatbul *et al.* 04]．こ

れらの研究はあくまでもセンサネットワークからデータを吸い上げる手法を研究しているだけであり，その吸い上げたデータを管理するシステムについては研究していない．

すなわち，**センサネットワークはセンサデータベースシステムのクライアント**だと考えられる．両者の関係を図 6.9 に示す．例えば Great Duck Island におけるセンサネットワークから得られるセンサデータは PostgreSQL に格納される [Mainwaring *et al.* 02]．これらのセンサデータには高鮮度化・周期的発信・時系列処理が要求されるだろうから，本研究で述べた KRAFT のようなシステムが求められると考えられる．

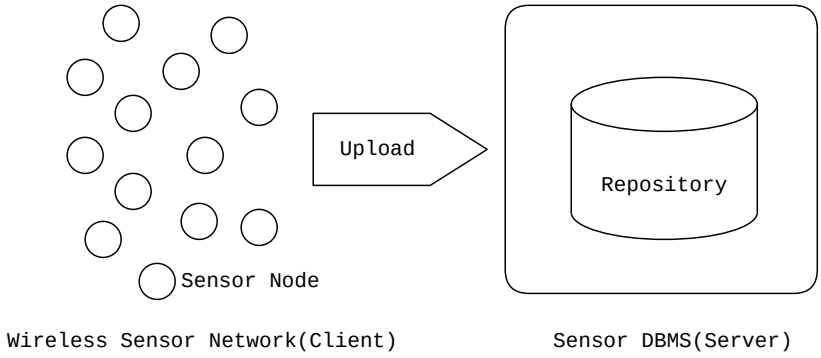


図 6.9: センサネットワークとセンサデータベースシステムの関係

6.5.　本章のまとめ

本章ではこれまでの論文内容に関して議論した．まず，どうしてセンサデータを即時永続化し，遅延評価をしないのか議論し，トランザクションを扱うデータベースシステムとしてはデータを即時永続化せざるを得ないことを述べた．次に，本論文の課題について様々に議論し，時系列データアノテーションの重要性について述べた．それから幅広い応用について議論した．最後にデータベースシステム開発論について議論し，容易なデータベースシステム開発技法がこれから益々重要になるだろうことを述べた．

第7章

結論と展望

7.1. 結論

センサ応用システムがデータベースシステムに要求する性質は，従来のデータベースシステムの機能に加えて，センサデータの高鮮度化，周期的発信，時系列処理，そして過負荷制御である．本論文では，センサデータの高鮮度化・周期的発信・時系列処理を実現するデータベースシステム KRAFT の実現方式を提案すると共に，従来は考えられて来なかった WAL の処理過程に着目した過負荷制御手法を提案した．

まず，センサデータを扱うアプリケーションを支援するデータベースシステムに関する従来研究は，高鮮度化，周期的発信，そして時系列処理の 3 方向から進められてきたと考えられる．従来研究では，センサデータをどうやってアプリケーションを使うか統一的な視点を持っていなかったために，部分的に有益な研究しかできなかった．それに対して，本論文は統一的な視点の下に，データベースシステムのコアに根ざす 3 つの問題を指摘し，それらを同時に解決するデータベースシステムの実現方式を示した．

次に，過負荷制御方式について，従来研究ではセンサデータの永続化処理過程に着目した研究は行われてこなかった．それに対して本研究はセンサデータをインプリサイス永続化することによる過負荷制御方式を提案し，専用実験システムにおいて評価した．

本論文の貢献は，センサデータを効率的に扱うためには，データベースシステムは新しい課題を解決する必要があることを示し，解決手法の一例を示したことにある．換言すれば，本論文はデータベースシステムがセンサを扱うためのパラダイムをデータベースコミュニティへ提示したと言えよう．

7.2. 展望

センサ応用システムはこれから爆発的に増加するだろう．センサデバイスと計算デバイスはこれからますます小型化し，大量生産され，それに伴い安価になろう．そ

して，センサというキーボードとは全く異なる入力インタフェースから得られるデータを扱うことにより，センサ応用システムはこれまでは人間にしかできなかった情報処理をますます実現するだろう．それに伴い，人間はコンピュータをより人間的に感じるようになるだろう．

センサデータの高鮮度化方式のために，ディスクが使われることはもはやあるまい．リモートメモリもしくは不揮発性メモリがその任を担うことになるだろう．周期的発信を受け取るシステムは，同時に実時間性も求めよう．実時間処理技法は，理論的に深く追求されていると同時に数多くの OS において成果が実現されている．現在はデータベースシステムへの応用はあまりないが，センサ応用システムの増加に伴い，データベースシステムへの応用されるようになると考えられる．そして対象とするセンサ応用システムの種類が増えるにつれて時系列処理手法はますます増えていかざるを得ない．時系列処理手法をデータベースシステムにおいて効率的に実現することはこれからますます重要になると考えられる．

PRIME Project[Anzai 94] を推進されていらした安西祐一郎先生から,「これからは**センサ**である」とのビジョンを 1998 年に頂いたことで本研究は開始された．あれから 7 年が経ち，安西先生のビジョンの通りにセンサ応用システムは発展したように思うが，残念なことにそれらを支援できるデータベースシステムは殆んどない．本論文はセンサというパラダイムをデータベースシステム研究に提示すると共にある程度の成果を示した．しかし本論文の成果はセンサ応用システムの支援という目標への橋頭堡に過ぎない．センサデータベースシステムというパラダイムにおいて提出される様々な課題に対して，今後とも基盤ソフトウェア研究者は挑まなければならないと私は考える．

謝辞

本論文のテーマを学部 4 年時にくださり，御指導を賜わってまいりました 慶應義塾大学理工学部教授　安西 祐一郎先生　に心より感謝いたします．理工学部長，塾長兼務の激務の中，安西先生は極めて本質的で豊かなアドバイスをくださったばかりではなく，全く研究の進まない自分を忍耐強く励まし続けてくださりました．また，研究のみならず社会常識についても御指導を賜わりましたことを心から感謝しております．

研究に関するあらゆることに対してつぶさに御指導を賜わりました 慶應義塾大学理工学部専任講師　今井 倫太先生　に心から御礼申し上げます．今井先生からは，研究姿勢，研究手法，そして論文執筆法まで事細かに懇切丁寧に教えて頂きました．今井先生がいらっしゃらなければ，自分は本論文を完成できなかったろうと思います．

データベースに関して御指導を賜わりました 慶應義塾大学理工学部専任講師　遠山 元道先生　に厚く御礼申し上げます．遠山先生には学部 4 年生の頃からデータベース研究を始めとしてあらゆる面で多くのご指導頂きました．遠山先生がいらっしゃらなければ，自分はデータベースに関して論文を書けなかったろうと思います．

データベース，コンピュータ，そして研究について素晴らしい御指導を賜わりました 慶應義塾大学環境情報学部教授　清木 康先生　に心から御礼申し上げます．清木先生には学会において親しくさせて頂いたばかりではなく，お忙しいなかディスカッションをさせて頂きましたり，はては合宿にも参加させて頂き，数多くの得がたい貴重な経験をさせて頂きました．清木先生にお会いしたときには常に励ましと元気を頂戴いたしまして，研究への情熱を湧き立てられました．

本論文に対し，大変有益なご助言を下さった 慶應義塾大学理工学部助教授　矢向 高弘先生　にお礼申し上げます．矢向先生に頂戴したアドバイスのおかげでデータベースシステムを大局的に捉え直すことができました．

安西・今井研究室秘書 永坂 弘子さんには日頃の研究室生活において大変お世話になりました．研究室で一緒に博士課程に進学した 梅澤 猛氏，大村 廉博士 (現 ATR)，宮崎 崇史氏とは苦楽を分かち合いました．研究室の先輩である白石 陽博士 (現 東大) にはデータベースに関して多くの有益な議論をして頂きました．

安西・今井研究室データベースグループの佐竹 聡氏，広瀬 健志郎氏，中村 学氏には忙しい中，本論文の校正をお手伝い頂きましたことに心から感謝します．彼ら

とは短い間でしたが基盤ソフトウェアに関する勉強や研究に関する意見交換を通じて誠に有益な勉強をさせて頂きました.

慶應義塾大学理工学部 遠山研究室の皆様には，折に触れて研究に関するご意見を頂戴したり，投稿論文の校正を手伝っていただきました．特に本論文の校正をお手伝い頂きました根本潤氏にお礼申し上げます.

トランザクションに関して幅広く深い勉強をさせて頂きました，　東京工業大学学術国際情報センター兼情報理工学研究科計算工学専攻 教授　横田 治夫先生　，奈良先端科学技術大学院大学情報科学研究科データベース学講座 助教授　宮崎 純先生　にお礼申し上げます．両先生には学会でトランザクションに関する勉強をさせて頂いたばかりではなく，ACM SIGMOD 日本支部 ネットワークトランザクション専門委員会で豊かな勉強をする機会を頂戴いたしました．周りにデータベースシステムに詳しい方がいらっしゃらなかったにも関わらず自分がデータベースシステムを開発できたのは，両先生に御指導を賜わったからであります.

未踏ソフトウェア創造事業を通じて御指導頂きました，　電気通信大学情報工学科 コンピュータ学講座 教授　竹内 郁雄先生　にお礼申し上げます．竹内先生にはIPA 未踏ソフトウェア創造事業を通して，ソフトウェア開発，研究者としての姿勢，そして自由な発想法につきまして豊かな御指導を賜わりました．竹内先生に頂戴した御指導がなければ KRAFT は決して完成しませんでした.

中学時代に御指導頂きました 東京工業大学経営工学専攻教授　藁谷 敏晴先生　にお礼申し上げます．藁谷先生には中学のころにゲーデルの不完全定理等の数学からバーンスタインの音楽までにわたり幅広く御指導を頂戴しました．自分が理工系に興味をもったのは藁谷先生に御指導を頂戴したからであります.

折に触れて御指導を賜わりました 慶應義塾大学情報工学科助教授　山崎 信行先生　にお礼申し上げます．山崎先生には輪講を通じてリアルタイムシステムに関する勉強をさせて頂きました.

最後になりますが，私のことを支え続けてくれた父 宏之，母 登喜恵，兄 俊之，姉 桃子，文子に深く感謝申し上げる次第であります．忙しい家業を手伝わずに自由に研究をさせてくれたが家族の理解なくば本論文は決して完成しませんでした.

2005 年 2 月
川島 英之

参考文献

[Abbott *et al.* 88] Robert K. Abbott, and Hector Garcia-Molina. Scheduling Real-time Transactions: a Performance Evaluation. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, pp. 1–12. Morgan Kaufmann Publishers, August 1988.

[Adelberg *et al.* 96] Brad Adelberg, Ben Kao, and Hector Garcia-Molina. Overview of the STanford Real-time Information Processor(STRIP). In *ACM SIGMOD Record*, Vol. 25, pp. 34–37, March 1996.

[Anderson *et al.* 91] Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, and Henry M. Levy. Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism. In *Proceedings of the 13th SOSP*, pp. 95–109, 1991.

[Anzai 94] Yuichiro Anzai. Human-Robot-Computer Interaction: A New Paradigm of Research in Robotics. *Advanced Robotics*, Vol. 8, No. 4, pp. 357–369, August 1994.

[Babcock *et al.* 02] Brian Babcock, Shivnath Babu, Mayur Datar, Rajeev Motwani, and Jennifer Widom. Models and Issues in Data Stream Systems. In *ACM Symposium on Principles of Database Systems*, 2002.

[Babcock *et al.* 03] Brian Babcock, Shivnath Babu, Mayur Datar, and Rajeev Motwani. Chain: Operator scheduling for memory mininization in data stream systems. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 253–264, 2003.

[Carney *et al.* 02] Don Carney, Ugur Cetinternel, Mitch Cherniack, Christian Convey, Sangdon Lee, Greg Seidman, Michael Stonebraker, Nesime Tatbul, and Stan Zdonik. Monitoring Streams - A New Class of Data Management Applications. In *Proceedings of the 28th International Conference on Very Large Data Bases*, 2002.

[Cha *et al.* 04] Sang K. Cha, and Changbin Song. P*TIME: Highly Scalable OLTP DBMS for Managing Update-Intensive Stream Workload. In *Proceedings of 30th International Conference on Very Large Data Bases*, pp. 1033–1044, 2004.

[Chandrasekaran *et al.* 03] Sirish Chandrasekaran, Owen Cooper, Amol Deshpande, Michael J. Franklin, Joseph M. Hellestein, Wei Hong, Sailesh Krishnamurthy, Sam Madden, Vijayshankar Raman, Fred Reiss, and Mehul Shah. TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In *Proceedings of the 2003 CIDR Conference*, 2003.

[Chandrasekaran *et al.* 04] Sirish Chandrasekaran, and Michael Franklin. Remembrance of Streams Past: Overload-Sensitive Management of Archived Streams. In *Proceedings of 30th International Conference on Very Large Data Bases*, pp. 348–359, 2004.

[Chen *et al.* 04] Shimin Chen, Anastassia Ailamaki, Phillip B. Gibbons, and Todd C. Mowry. Improving Hash Join Performance through Prefetching. In *Proceedings of the IEEE International Conference on Data Engineering*, pp. 116–127, 2004.

[Date 84] C. J. Date. データベース・システム概論, 第 1 章. 丸善株式会社, 第 3 版, 1984.

[Date 97] C. J. Date. データベース・システム概論, 第 13 章, p. 411. 丸善株式会社, 第 6 版, 1997.

[Datta *et al.* 97] Anindya Datta, and Igor R. Viguier. Providing Real-Time Responce, State Recency and Temporal Consistency in Databases for Rapidly Changing Environments. *Information Systems*, Vol. 22, No. 4, pp. 171–198, 1997.

[Gray *et al.* 01a] Jim Gray, and Andreas Reuter. トランザクション処理，概念と技法, 第 13 章. 日経 BP, 2001.

[Gray *et al.* 01b] Jim Gray, and Andreas Reuter. トランザクション処理，概念と技法, 第 3 章. 日経 BP, 2001.

[Härder *et al.* 83] Theo Härder, and Andreas Reuter. Principles of Transactions Oriented Database Recovery. *ACM Computing Surveys*, Vol. 15, No. 4, pp. 287–317, 1983.

[Hvasshovd *et al.* 95] Svein-Olaf Hvasshovd, Øystein Torbjørnsen, Svein Erik Bratsberg, and Per Holager. The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response. In *Proceedings of the 21th International Conference on Very Large Data Bases*, pp. 469–477, September 1995.

[IDC 03] IDC. http://www.idc.com/getdoc.jsp?containerId=pr2004_06_03_112013, 2003.

[Informix 97] Informix. Informix TimeSeries DataBlade Module User's Guide, April 1997.

[Kanda *et al.* 02] Takayuki Kanda, Hiroshi Ishiguro, Tetsuo Ono, Michita Imai, and Ryohei Nakatsu. Development and Evaluation of an Interactive Humanoid Robot "Robovie". In *Proceedings of IEEE International Conference On Robotics and Automation*, pp. 1848–1855, 2002.

[Kang *et al.* 04] Kyoung-Don Kang, Sang H. Son, and John A. Stankovic. Managing Deadline Miss Ratio and Sensor Data Freshness in Real-Time Databases. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 16, No. 10, pp. 491–502, October 2004.

[Katsura *et al.* 04] Seiichiro Katsura, and Kouhei Ohnishi. Human Cooperative Wheelchair for Haptic Interaction Based on Dual Compliance Control. *IEEE Transactions on Industrial Electronics*, Vol. 51, No. 1, February 2004.

[Keogh *et al.* 99] Eamonn J. Keogh, and Michael J. Pazzani. Scaling up Dynamic Time Warping for Datamining Applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 285–289, 1999.

[Lee *et al.* 01] Juchang Lee, Kihong Kim, and Sang K. Cha. Differential Logging: A Commutative and Associative Logging Scheme for Highly Parallel Main Memory Databases. In *Proceedings of the IEEE International Conference on Data Engineering*, pp. 173–182, 2001.

[Lehr *et al.* 95] Matthew R. Lehr, Young-Kuk Kim, and Sang H. Son. Managing Contention and Timing Constraints in a Real-Time Database System. In *Proceedings of the IEEE Real-Time Systems Symposium*, pp. 332–341, 1995.

[Lin *et al.* 87] Kwei-Jay Lin, Swami Natarajan, and Jane W.-S.Liu. Concord: A System of Imprecise Computation. In *Proceedings of the Eleventh Annual International Computer Software and Applications Conference (COMPSAC 87)*, pp. 75–81, 1987.

[Madden *et al.* 03] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. The Design of an Acquisitional Query Processor for Sensor Networks. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 491–502, 2003.

[Mainwaring *et al.* 02] Alan Mainwaring, Joseph Polastre, Robert Szewczyk, David Culler, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applicatoins*, pp. 88–97, 2002.

[Mohan 99] C. Mohan. Repeating History Beyond ARIES. In *Proceedings of 25th International Conference on Very Large Data Bases*, pp. 1–17, 1999.

[Shao *et al.* 04] Minglong Shao, Jiri Schindler, Steven W. Schlosser, Anastassia Ailamaki, and Gregory R. Ganger. Clotho: Decoupling Memory Page Layout from Storage Organization. In *Proceedings of the Fourteenth International Conference on Very Large Data Bases*, pp. 696–707. Morgan Kaufmann Publishers, August 2004.

[Silbershatz *et al.* 02] Avi Silbershatz, Peter Baer Galvin, and Greg Gagne. *Operating System Concepts*, chapter 16, p. 581. John Wiley & Sons, Inc, 6th edition, 2002.

[Tatbul *et al.* 04] Nesime Tatbul, Mark Buller, Reed Hoyt, Steve Mullen, and Stan Zdonik. Confidence-based Data Management for Personal Area Sensor Networks. In *Proceedings of the First ACM International Workshop on Wireless Sensor Networks and Applicatoins*, pp. 88–97, 2004.

[Vlachos *et al.* 03] Michail Vlachos, Marios Hadijieleftheriou, Dimitrios Gunopulos, and Eamonn Keogh. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 216–225, August 2003.

[Vrbsky *et al.* 93] Susan V. Vrbsky, and Jane W. S. Liu. APPROXIMATE – A Query Processor That Produces Monotonically Improving Approximate Answers. *IEEE Transactions on Knowledge and Data Engineering*, Vol. 5, No. 6, pp. 1056–1068, 1993.

[XBOW 04] XBOW. http://www.xbow.com, 2004.

[宮本 他 01] 宮本真理子, 池田高志, 岡田健一. 無線 LAN 環境におけるプレゼンテーションのためのマルチキャストプロトコル. 情報処理学会論文誌, Vol. 42, No. 12, pp. 3903–3101, 2001.

[高根 80] 高根芳雄. 多次元尺度法, 第 2 章. 東京大学出版会, 1980.

[小杉 他 04] 小杉尚子, 櫻井保志, 森本正志. ハミング検索のための音楽データ自動時間正規化手法. 情報処理学会論文誌データベース, Vol. 55, No. SIG7(TOD22), pp. 163–177, June 2004.

[赤松 03] 赤松幹之. 運転行動データベースの構築とアクティブセーフティ技術への利用. 自動車技術, Vol. 57, No. 12, pp. 34–39, 2003.

[大崎 他 99] 大崎竜太, 上原邦昭. DTW を用いた身体動作における基本動作の抽出. 情報処理学会研究報告データベースシステム, Vol. 119-47, pp. 279–284, July 1999.

[中井 他 00] 中井敏久, 佐藤範之. 移動通信環境におけるインターネットビデオ-リアルタイムアプリケーションを可能とするプロトコルの提案-. 沖電気研究開発, Vol. 67, No. 1, pp. 53–56, 2000.

[東大地震研 05] 東大地震研. http://wwweic.eri.u-tokyo.ac.jp/mirror/, 2005.

[氷山 04] 氷山素子. デーモン君のソース探検, 第 16 章. 株式会社アスキー, 初版, 2004.

[本田 03] 本田喜久. 人–ロボットコミュニケーションにおける相互適応メカニズムの実現. 慶應義塾大学大学院理工学研究科開放環境科学専攻コンピュータ科学専修，修士論文, 2003.

論文目録

【 主論文に関する公刊論文 】

1. 川島 英之，遠山 元道，今井 倫太，安西 祐一郎，“リモートメモリを用いたセンサデータストリームの永続化”，情報処理学会論文誌：データベース．Vol.44， No.SIG12(IPSJ Transactions on Databases 19)， pp.98-109 (2003年 9 月).

2. 川島 英之，今井 倫太，遠山 元道，安西 祐一郎，“センサデータベースシステム KRAFT の設計と実装”，情報処理学会論文誌：データベース．Vol.45，No.SIG14(IPSJ Transactions on Databases 24)， pp.39-53 (2004 年 12 月).

【 国際会議発表 】

3. Hideyuki Kawashima, Motomichi Toyama, Michita Imai, Yuiciro Anzai, “Providing Persistence for Sensor Data Stream with Temporal Consistency Conscious WAL” Proceedings of IASTED International Conference on Information Systems and Databases(ISDB 2002), pp.13–18, September 2002.

4. Hideyuki Kawashima, Motomichi Toyama, Michita Imai, Yuiciro Anzai, “Providing Persistence for Sensor Stream with Light Neighbor WAL” Proceedings of Pacific Rim International Symposium on Dependable Computing(PRDC 2002), pp.257–264，December 2002.

5. Hideyuki Kawashima, “Development of a Sensor Database System for Communication Robots” In Proceedings volume 2 of the 21st Annual British National Conference on Databases pp.130–131，July 2004.

6. Hideyuki Kawashima, Michita Imai, Motomichi Toyama, Yuiciro Anzai, “Improving Freshness of Sensor Data on KRAFT Sensor Database System”,

Proceedings of Tenth International Workshop on Multimedia Information Systems. pp.20–29，August 2004.

【 国内学会発表 】

7. 川島英之　「センサデータベース管理システムの開発」　第 44 回プログラミング・シンポジウム　2003 年 1 月 8 日〜10 日　神奈川県足柄下郡箱根　箱根ホテル小桶園

8. 川島英之，今井倫太，遠山元道，安西祐一郎「センサネットワーク用データベースシステムの提案」 第 1 回センサネットワーク研究会 pp.51-58 2004 年 12 月 9 日〜10 日，東京電機大学神田キャンパス工学部 7 号館 1F 丹羽ホール

【 同一著者による他の研究成果 】

9. 川島英之，嶋田総太郎，安西祐一郎 「効率的な時系列データ探索手法」　電子情報通信学会データベースシステム研究会　第 10 回データ工学ワークショップ (DEWS99) オンライン出版　1999 年 3 月 4 日〜6 日 鹿児島県指宿市休暇村鹿児島サン・オーシャン・リゾート

10. 川島英之，遠山元道，安西祐一郎「分散時系列データベースにおける問い合わせ処理の QoS 保証に関する研究」 情報処理学会「システムソフトウェアとオペレーティング・システム」No.84-10　2000 年 5 月 25 日〜26 日沖縄県恩納村　ホテルムーンビーチ沖縄

11. 川島英之，遠山元道，安西祐一郎 「Multi-level Virtual Deadline: Temporal Consistency と Real-Time を同時に満足する資源割り当て手法」　データベースと Web 情報システムに関する IPSJ DBS/ACM SIGMOD Japan Chapter/JSPS-RFTF AMCP 合同シンポジウム　pp.105-112　2000 年 12 月 6 日〜8 日　東京都港区台場フジテレビジョン 1 F マルチシアター/ホテル日航東京会議室 アポロン

12. 川島英之，遠山元道，安西祐一郎　「リアルタイムデータベースのためのデータ品質管理手法の提案」　実時間処理に関するワークショップ (RTP2001)/情報処理学会「システムソフトウェアとオペレーティング・システム研究会」合同ワークショップ No.86-13　pp.75-82　2001 年 3 月 5 日〜6 日　奈良先端科学技術大学院大学情報科学研究科大会議室 L1

13. 川島英之，遠山元道，安西祐一郎 「センサデータに対する実時間検索を支援する機構」　第 63 回（平成 13 年後期）情報処理学会全国大会　pp.175-176　2001 年 9 月 26 日〜28 日　山口大学吉田地区キャンパス

14. 川島英之，遠山元道，安西祐一郎 「メモリロギングによるセンサデータ挿入処理の高速化」　電子情報通信学会　第 13 回データ工学ワークショップ（DEWS2002）　オンライン出版 2002 年 3 月 4 日〜6 日　岡山県倉敷市 倉敷国際ホテル

15. 川島英之，遠山元道，今井倫太，安西祐一郎　「波形特徴を用いた類似シーケンス検索」 第 128 回情報処理学会データベースシステム研究会 Vol.2002 − DBS-128　pp.529-534　2002 年 7 月 17 日〜19 日　栃木県日光国立公園・鬼怒川温泉あさやホテル

16. 川島英之，遠山元道，今井倫太，安西祐一郎　「データ鮮度と実時間応答を考慮したスケジューリング方式」　電子情報通信学会データ工学研究専門委員会＆電子情報通信学会ディペンダブルコンピューティング研究専門委員会　インターネット環境でのデータ工学とディペンダビリティに関する研究会　Vol. 2002 − 85〜89　pp.25-30　2002 年 10 月 17 日〜18 日　千葉県千葉市・IBM 幕張事業所 602 教室

17. 見山成志，川島英之，今井倫太，安西祐一郎 「METIS：無線環境を考慮したトランザクション処理システム」　第 128 回情報処理学会データベースシステム研究会 Vol.2002 − DBS-128　pp.17-24　2002 年 7 月 17 日〜19 日　栃木県日光国立公園・鬼怒川温泉あさやホテル

18. 本田喜久，川島英之，今井倫太，安西祐一郎　「CSS-tree の挿入処理の高速化」　第 128 回情報処理学会データベースシステム研究会　Vol.2002 − DBS-128　pp.59-66　2002 年 7 月 17 日〜19 日　栃木県日光国立公園・鬼怒川温泉あさやホテル

19. Satoru Satake, Hideyuki Kawashima, Michita Imai, "LACOS: CONTEXT MANAGEMENT SYSTEM FOR A SENSOR-ROBOT NETWORK," The 10th IASTED International Conference on Robotics and Applications (RA 2004), pp. 160-165, August 23-25, 2004, Sheraton Waikiki Hotel, Honolulu, Hawaii, USA

20. Satoru Satake, Hideyuki Kawashima, Michita Imai, "CDBMS:Database Management System for a Communication Robot," 12th IEEE Workshop Robot and Human Interactive Communication (RO-MAN 2003),

pp.205-210, October 31 - November 2, 2003, Westin SanFrancisco Airport Hotel, Millbrae, California, USA

21. Kenshiro Hirose, Satoru Satake, Hideyuki Kawashima, Michita Imai, and Yuichiro Anzai "Development of communication contents description language" IEEE SMC 2004, pp.2896-2900 2004 年 10 月 10 日〜10 月 13 日 Netherland Congres Center, The Hague, Netherland

22. 佐竹聡，川島英之，今井倫太「ブラウジングロボット: コミュニケーションロボットによるインターネットコンテンツの閲覧」第 136 回情報処理学会知能と複雑系研究会 & 第 65 回人工知能学会知識ベース研究会合同研究会, pp. 49-55, 2004 年 8 月 4-6 日 長野県 メゾン軽井沢

23. 佐竹聡，川島英之，今井倫太「データベースを用いたコミュニケーションロボットシステムの構築」電子情報通信学会 信学技法, Vol.103, No.32,pp.7-12 電子情報通信学会 ヒューマンコミュニケーション研究会映像情報メディア学会メディア工学研究会 2003 年 4 月 25 日 早稲田大学理工学部大久保キャンバス

24. 佐竹聡，川島英之，今井倫太「データベースを用いたコミュニケーションロボットシステムの構築」第 65 回情報処理学会全国大会, No.2, pp.417-418 2003 年 3 月 25-27 日 東京工科大学八王子キャンパス

25. 広瀬健志郎, 佐竹聡, 川島英之, 今井倫太「コミュニケーションロボットによる WWW 上のコンテンツ閲覧へ向けたモーション記述言語の設計と実装」情報処理学会第 66 回全国大会,No. 3, pp.151-152 2004 年 3 月 9 日〜12 日 神奈川県慶應義塾大学湘南藤沢キャンパス

26. 広瀬健志郎, 佐竹聡, 川島英之， 今井倫太「異種ヒューマノイド間でのコミュニケーションコンテンツの共有の実現」 第 110 回ヒューマンインターフェース研究発表会 pp.47-52 2004 年 9 月 10 日，京都府国際電気通信基礎技術研究所 (ATR)