システムソフトウェアと
オペレーティング・システム 84−10
（2000．5．25）

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

○川島 英之　　遠山 元道　　安西 祐一郎

慶應義塾大学 計算機科学専攻

〒223-8522 横浜市港北区日吉 3-14-1

E-mail: {kawasima@ayu,toyama@db,anzai@ayu}.ics.keio.ac.jp

概要 近年のセンサとコンピュータの発達により、実世界の情報をオンラインで取得しリアルタイムに応答するアプリケーションが現れてきた。そのようなサービスを提供するリアルタイム時系列データベースはTemporal ConsistencyとReal-Timenessの2つの要求を満たすべき。しかし計算資源が有限であるために、これらを同時に満たすことは困難である。そこで本研究ではVirtual Deadlineに基づく計算資源管理アルゴリズムを設計し、分散時系列データベースに実装することにより、上記の2つの要求を同時に満足させることを試みた。設計したアルゴリズムが分散環境において有効であることを確かめるために、分散環境において時系列データベースのプロトタイプを実装した。

A Research about QoS Guarantee for Query Processing on Distributed Time-Series Database

Hideyuki Kawashima　Motomichi Toyama　Yuichiro Anzai

Department of Computer Science, Keio University
3-14-1 Hiyoshi, Kouhoku-ku, Yokohama, Kanagawa 223-8522 Japan

Abstract By the development of sensors and computers, new applications which needs both fresh image of read-world and real-time response are appeared. For such applications, database should satisfy Temporal Consistency and Real-Timeness. But that is diffcult by the limitation of computational resources. So, we designed an algorithm based on Virtual Deadline to satisfy Temporal Consistency and Real-Timeness at the same time on time-series database. We implemented the algorithm on a prototype time-series database in distributed environments, and evaluating the designed algorithm.

1. はじめに

センサとコンピュータの発達により、実世界の情報がオンラインで取得され、リアルタイムに処理されるようになってきた。

この流れに対応すべく、筆者の所属する研究室でもPRIME Project[1]の一環としてセンサネットワーク[2]の研究をおこなってきた。

センサネットワーク研究の目的は、環境内に張り巡らされたセンサにより取得した環境情報をもとに、生活環境を快適にそして、安全にするための基盤技術を確立することである。

物理的環境情報を取得、管理するために、センサデータを取得するセンサノードや、計算機ネットワークからのセンサネットワーク利用を可能にするセンサノードマネージャ[3]、ノードの物理的位置情報を取得するIPS(IndoorPositioningSystem)[4] などが研究されてきた。

また、それらの環境情報を利用する機構としてモバイルエージェント [5][6]、ディレクトリサービス [7] などが構築されてきた。

しかし、環境情報を取得する機構と利用する機構の間には、環境情報を管理し、アプリケーションに提供する機構である、データベースが存在しなかった。

さて、このようなデータベースには 2 種類の要求が発生する。ひとつは実世界の状態とデータベースの状態を近づけるために、時系列で発生するセンサ情報をデータベースに蓄えることである。そして、もうひとつはデータベースクライアントにより発行されたトランザクションを処理することである。

そこで本研究では、それらを満たそうとするアルゴリズムを設計し、そのアルゴリズムを分散時系列データベースのプロトタイプに実装し、評価する。

2. リアルタイム時系列データベース

時間の変化に伴い新しいデータの挿入が行われるデータベースを時系列データベースと定義する。

一方、時系列データベースには時々刻々と新しいデータが入力されるので、クライアントは鮮度が落ちない内にデータを読むことを求める。

このような、時系列データに対するリアルタイムな応答を提供するシステムを、本研究ではリアルタイム時系列データベースと定義する。

リアルタイム時系列データベースには 2 種類の要求が発生する。1) ひとつは時系列で発生するセンサ情報をデータベースに蓄え、実世界の状態とデータベースの状態を近づけることであり、2) もうひとつはデータベースクライアントにより発行されたトランザクションをデッドライン以内に処理することである。

しかし、これらの要求を同時に満たすことは困難である。なぜならば、計算機はある瞬間にひとつの処理しか実行できないからである。

[8] ではアルゴリズム **Virtual Deadline of Update** が紹介されており、データ取得を Virtual Deadline と呼ばれる時間だけ経過した後に初めて許可することにより、前述の 2 つの要求を満たそうとしている。

しかし [8] ではシミュレーションによる評価しかなされておらず、データ構造とタスクモデルの詳細が述べられていない。

さらに、[8] ではネットワークを介した分散化を無視しているので、ネットワークによる遅延及び分散化によるオーバヘッドが考慮されていない。早晩、リアルタイム時系列データベースがネットワークでつながれた分散環境に適用されるだろうことを考慮すると、リアルタイム時系列データベースの研究は分散環境を視野にいれて行うことが望ましい。

そこで本研究ではネットワークでつながれた分散環境において Virtual Deadline を用いたリアルタイム時系列データベースのプロトタイプを実装した。そしてクライアントへ供給できたデータの品質とトランザクション応答時間に関して評価を行っている。

本研究ではリアルタイム時系列データベースがクライアントに供給するデータの品質を以下の 2 点について評価している。

- Temporal Consistency

  クライアントにより、時刻 t における複数のデータを要求されたとき、それらのデータが蓄えられた時刻は t から許容限度内に存在するか？

- Real-Timeness

クライアントが発行したデータ読み出しがデッドラインまでに終了するか？

3. システムモデル

本節では本研究で扱うデータおよびタスクを定義する。

3.1. データモデル

時系列データベースでは時系列で入力されるデータを扱う。それゆえ各エンティティは多数のヴァージョンを保持する。

そこで本論文ではデータを以下のように定義する。

- **DATAITEM**

  実世界で時系列に情報を発信するものを時系列データベースにマッピングしたオブジェクトを **DATAITEM** と定義する。

- **VERSIONDATA**

  実世界でそれが発生した時刻を属性として持つデータを **VERSIONDATA** と定義する。

DATAITEM 及び **VERSIONDATA** の関係を図1に示す。

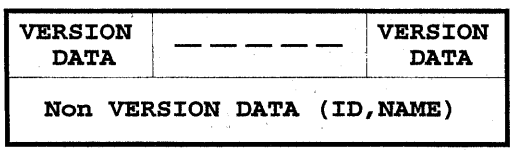

図 1: データモデル

以降、本論文では **DATAITEM** i を D_i とし、**DATAITEM** D_i の **VERSION DATA** v を D_i^v として表す。

3.2. タスクモデル

次に、本研究で扱うタスクの定義を行う。

- **INSERTION**

  センササーバがデータベースに蓄えられている D_i に D_i^v を挿入する手続きを **INSERTION** と定義する。[8] ではこの手続きを **UPDATE** と定義しているが、通常データベースにおいて **UPDATE** はデータ更新を表す。誤解を避けるために本研究では **INSERTION** を用いる。

- **TRANSACTION**

  クライアントがデータベースに蓄えられている、ある時刻の複数項目のデータを読み込む手続きを **TRANSACTION** と定義する。通常データベースの研究における **TRANSACTION** とは ACID 特性を持つデータベースへの一連の操作を表すが、本論文ではこれ以降 **TRANSACTION** を上の定義で用いる。

- **READ**

  一つの D_i^v を読み込む手続きを **READ** と定義する。

  TRANSACTION は単数もしくは複数の **READ** によって構成される。

4. Virtual Deadline

本節では Temporal Consistency と Real-Timeness の両方を保つためのアルゴリズム Virtual Deadline[8] について述べる。

Virtual Deadline のアルゴリズムは次のとおりである。

> *Algorithm* **Virtual Deadline**
>
> 時系列データベースにオペレーション **INSERTION** が入ったとき、時刻が Virtual Deadline に達していなかったならばそれを破棄する。さもなくばそれを実行する。

Virtual Deadline 以前に時系列データベースに入力された **INSERTION** は破棄される。

それゆえ Temporal Consistency が満たされにくいときには、**INSERTION** を成功させる確率を増やすために Virtual Deadline を引き

下げる。これにより **INSERTION** が成功する確率が上昇する。しかしこのとき **TRANSACTION** に割り当てられる計算資源が少なくなる。

逆に Real-Timeness が満たされにくいときには、**TRANSACTION** に割り当てる計算資源を増やすために、Virtual Deadline を引き上げて **TRANSACTION** の成功確率を高める。ただし、**INSERTION** 成功確率は減る。

図 2に Virtual Deadline の概念図を示す。

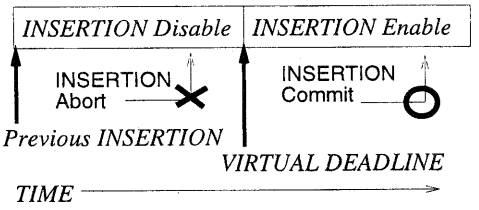

図 2: Virtual Deadline の概念図

4.1. Temporal Consistency

TRANSACTION が時刻 t における複数の **VERSIONDATA**、すなわち D_i^v、D_j^w を求めたとき、D_i^v、D_j^w の時間的距離が近いならば D_i^v と D_j^w は Temporal Consistency を満たすとする。

4.1.1. Perfect Temporal Consistency

さて、Temporal Consistency を完全に満たすには時刻 t_j において D_i と D_j の値が要求されたら正確に時刻 t_j の値を供給すれば良い。

しかしこれを満たすためにはあらゆる **INSERTION** を破棄することなく実行しなければならない。しかし限られた計算資源を Real-Timeness を満足させるためにも使用しなければならないために、その実現は困難である。

4.1.2. Reasonable Temporal Consistency

そこで reasonable な Temporal Consistency を満たすことを考える。D_i^v が発生した時刻を $TS(D_i^v)$(ここで TS は *TimeStamp* を表す) とし、**READ** が要求した時刻と $TS(D_i^v)$ 間の許容できる時間間隔を遅延限界 **Maximum Tolerable Delay(MTD)** とする。

READ が t_1 において D_i と D_j の値を要求したが、Perfect Temporal Consistency を満たすことができなければ t_2 における D_i の値と t_3 における D_j の値を供給する。ただし t_1,t_2,t_3 の時間的距離はかなり近いものだとする。

このとき D_i^v と D_j^w について

$$|TS(D_i^v) - TS(D_j^w)| \leq MTD$$

が成り立つならば Temporal Consistent だとみなす。

ここで次のようなアルゴリズムを用いて Reasonable Temporal Consistency を満たすよう試みる。

READ が D_i^v を読んだとき、それが要求時刻から MTD 以内のものであるか否か調べ、もしそれが MTD よりも離れていたならば D_i の Virtual Deadline を増やして **INSERTION** へ割り当てる計算資源を増やす。

Request TimeStamp of D_i(**READ** に要求された D_i の時刻) を $RQTS(D_i)$ としたとき、次式が成り立つ。

if $(|D_i^v - RQTS(D_i)| \leq MTD)$
then $\alpha \leftarrow \alpha \times \beta \quad (\beta > 1)$
$VD \leftarrow \alpha \times VD$
$(VD : VirtualDeadline)$

4.2. Real Timeness を保つためのアルゴリズム

TRANSACTION 終了時刻がデッドラインを越えていたならば、**TRANSACTION** を構成する **READ** が参照した D_i の Virtual Deadline を減らし、**READ** へ割り当てる計算資源を増やす。

if (**TRANSACTION** 終了時刻 > デッドライン)
  then $\alpha \leftarrow \alpha \times \gamma \quad (\gamma < 1)$
$VD \leftarrow \alpha \times VD$
$(VD : VirtualDeadline)$

4.3. 関連研究

Virtual Deadline は [8] で提案されたアルゴリズムである。しかし [8] における Virtual Deadline の評価にはいくつかの問題点が存在する。まず、[8] ではシミュレータを用いて Virtual Deadline を評価しているために、データベースのデータ構造、システムで発生するタスクが不明確であるため、実装にはアイデアが必要となる。2つ目に、[8] ではネットワークを考慮していない。3つ目に、[8] ではアルゴリズムに無駄がある。提案されている Virtual Deadline の変更手法では、Temporal Consistency、Real-Timeness に違反しようとしまいと、Virtual Deadline を変更している。違反をしない場合に Virtual Deadline を変更することは計算資源の浪費である。

5. 設計

そこで本研究では上記の問題を解決するために、計算資源割り当て方策に Virtual Deadline を用いるリアルタイム時系列データベースのプロトタイプを分散環境において設計、実装した。本節ではプロトタイプの設計について述べる。

5.1. システム設計の概要

クライアントは分散リアルタイム時系列データベースへデータを要求するために Transaction Processing Server と接続し、Data Request を発行する。

Transaction Processing Server はクライアントが発行した Data Request を満たすために、クライアントからの要求に応じて単数もしくは複数のネットワークを介した Time-Series Database へ対して **TRANSACTION** を発行する。

Transaction Processing Server から依頼を受けた Time-Series Database は **TRANSACTION** を実行し、その結果を Transaction Processing Server に返す。

Transaction Processing Server は発行した全ての **TRANSACTION** の結果が与えられるまで待ち、結果が揃ったらそれをクライアントへ返す。

本システムの設計図を図 3に示す。

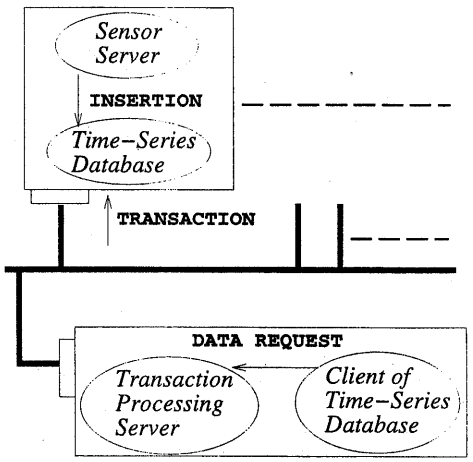

図 3: 分散時系列データベースの設計図

以降、3分散時系列データベースのソフトウェアコンポーネントについて述べていく。

5.2. Time-Series Database

ソフトウェアコンポーネント Time-Series Database のデータ構造およびタスク処理手続きの設計について述べる。

5.2.1. データ構造

DATAITEM D_i を以下の属性をもつ構造体として設計した。

```
struct DATAITEM {
  DATAITEM 識別子;
  DATAITEM の名前;
  最新 VERSIONDATA を表す添字;
  VERSIONDATA 配列;
  DATAITEM の mutex;
};
```

そして **VERSION DATA** D_i^v を以下の属性をもつ構造体として設計した。

```
struct VERSIONDATA {
  DATAITEM 識別子;
  データ;
  データが生成された時刻;
};
```

5.2.2. タスク処理

Time-Series Database は 2 種類のタスク処理を行う。Time-Series Database へ **VERSIONDATA** を挿入する処理 **INSERTION** と、Transaction Processing Server に要求された **VERSIONDATA** を返却する手続き **TRANSACTION** である。

- **INSERTION** 処理アルゴリズム

  1. Sensor Server から **VERSIONDATA** の到着を待つ。
  2. 到着した **VERSIONDATA** をバッファへ入れる。
  3. **VERSIONDATA** の Virtual Deadline を id をキーに検索、代入する。
  4. バッファがいっぱいならば次へ、さもなくば初めに戻る。
  5. バッファ内 **VERSIONDATA** を Virtual Deadline 順に整列する。
  6. 時刻がバッファ先頭の **VERSIONDATA** の Virtual Deadline を越えたら、該当する **DATAITEM** を lock し、書き込みを行う。
  7. lock していた **DATAITEM** を unlock する。

- **TRANSACTION** 処理アルゴリズム

  1. Transaction Processing Server からの **TRANSACTION** 到着を待つ。
  2. **READ** オペレーションを揃える。
  3. **TRANSACTION** を構成する、すべての **READ** について lock をかける。これは 2 相ロックの lock phase に該当する。
  4. **TRANSACTION** を構成する、すべての **READ** について、読み込みを要求するデータを返却バッファに入れる。
  5. **TRANSACTION** を構成する、すべての **READ** について unlock をかける。これは 2 相ロックの unlock phase に該当する。
  6. Time-Series Database は、**TRANSACTION** がアクセスした全てのデータについて Temporal Consistency を調べる。もしもそれが満たされなかったならば、Virtual Deadline を引き下げる。
  7. 返却バッファのデータを Transaction Processing Server へ返す。
  8. データを返された Transaction Processing Server は、**TRANSACTION** の返却時刻がデッドラインを越えていないか判定し、越えている場合は Time-Series Database に対して、**TRANSACTION** がアクセスした全 **DATAITEM** の Virtual Deadline を引き上げるよう要請する。

5.3. Sensor Server

Sensor Server は Time-Series Database に定期的にデータを供給するソフトウェアコンポーネントである。

5.4. Transaction Processing Server

Transaction Processing Server は Time-Series Database へ **TRANSACTION** を発行し、**TRANSACTION** が返却したデータを Client へ渡すソフトウェアコンポーネントである。

5.5. Client

Client は Transaction Processing Server に対して単数もしくは複数の **TRANSACTION** の実行を依頼し、そのサービスを享受する。

6. 実装環境

表1に示した計算機上に、分散リアルタイム時系列データベースのプロトタイプを実装した。計算機間をつなぐネットワークには 100Mbps Ethernet を用いた。そして 2 相ロックの実現には、pthread を利用した。

CPU	OS
Sun-UltraSPARC-IIi 333MHz	Solaris 2.6
Sun-UltraSPARC 167MHz	Solaris 2.6
Sun-UltraSPARC-IIi 440MHz	Solaris 2.6

表 1: 実験に用いた計算機

7. 評価

上記の環境において、Temporal Consistency ミス率及び Real-Timeness ミス率について評価をおこなっている。

デッドライン、MTD、**INSERTION** 周期を一定とし、「現時点に最も近い時間について、全 **DATAITEM** を教えてください」と問い合わせを掛ける。

横軸に **TRANSACTION**/sec、縦軸に Miss 率をとり、結果をグラフを描いたとき、Virtual Deadline を用いた場合のグラフは、用いない場合のグラフに比べて振動が少ないだろうと推測している。推測図を図 4に示す。

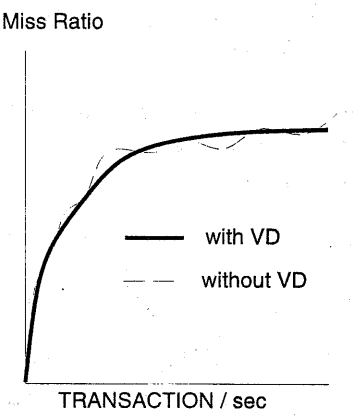

図 4: 結果予想図

8. 課題

今回の研究ではリアルタイム性を保証する機構を整えることができなかった。今後、以下の項目を検討し、分散時系列データベースにおける QoS 保証に関する研究をすすめていく予定である。

- リアルタイム OS の使用
- ネットワーク帯域保証
- 中間ノードにおけるキュー内のパケット優先度制御
- ディスクアクセス

9. まとめ

本研究ではリアルタイム時系列データベースにおいて要求される 2 つの性質である Temporal Consistency と Real-Timeness を同時に満たすために、Virtual Deadline を基礎としたアルゴリズムを設計し、それを分散環境におけるプロトタイプシステムに実装した。

参考文献

[1] Y. Anzai. Human-Robot-Computer Interaction: A New Paradigm of Research

in Robotics. *Advanced Robotics*, Vol. 8, No. 4, August 1994.

[2] M.Kohno. *An Addressing Scheme for Dynamic Sensor Networks.* PhD thesis, Keio University, March 1999.

[3] 大村, 河野, 安西. 移動センサノードに対応したセンサノード管理システムの拡張. 情報処理学会研究報告, Vol. 99, No. 32, 1999.

[4] 松永, 河野, 安西. 超音波による室内オブジェクトの 3 次元位置検出システムの設計と実装. 第 15 回日本ロボット学会学術講演会予稿集, 第 1 巻, 1997.

[5] 梅澤, 佐藤, 安西. 移動エージェントによるユーザ追跡型サービス. 第 8 回マルチエージェントと協調計算ワークショップ, 1999.

[6] 谷澤, 佐藤, 梅澤, 安西. モバイルエージェントによる移動ユーザへのサービス実現. 情報処理学会第 60 回全国大会, 第 3 巻, March 2000.

[7] 白井, 桜井, 鈴木, 安西. ユーザとサービスの位置関係に基づいた lookup 機構の設計と実装. In *DICOMO2000, accepted*, 2000.

[8] 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.