センサデータベース管理システムの開発

Development of a Sensor Database Management System

川島 英之
KAWASHIMA Hideyuki

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
〒 223-8522 神奈川県横浜市港北区日吉 3-14-1
E-mail: kawasima@ayu.ics.keio.ac.jp

ABSTRACT. This paper describes the details of a sensor database management system. The features of the developped system are: fast write ahead logging, similar sequence retrieval based on euclid distance, and multi-thread index manager.

1 背景

近年、センサデータを扱うアプリケーションが現れてきている。例えば、リアルタイム地震防災システム (SUPREME)、リアルタイム音楽伴奏システム、ロボット状況依存処理、そしてスポーツの技量測定である。

リアルタイム地震防災システム

1 番目のアプリケーションはリアルタイム地震防災システムである。リアルタイム地震防災システムの実例には、東京ガスが 2001 年末から稼働を開始した SUPREME がある。SUPREME の目的はガス管の遠隔遮断システムだが、これを地震センサネットワークとして使えば、細かい地域に応じた震度情報が得られるため、地震のスピードや進行方向などを把握できるようになり、地震発生時に、無数のセンサによって震源や規模を瞬時に特定し、地震の揺れが及ぶであろう地域を予想できるようになる。地震が伝わるスピードより光ファイバ通信のスピードの方が速いので、数秒早く地震の到来を通知できるようになる。この場合には、複数の地震センサデータを一括して表示/通達する必要があるために、複数系列のセンサデータを一括管理できるデータベース管理システムが必要になる。

リアルタイム伴奏システム

2 番目のアプリケーションは、ヴァイオリンを演奏しているときに、演奏速度と音程に合わせてオーケストラの音を出してくれるヴァイオリン伴奏システムである。このシステムは演奏者の演奏テンポが速まれば、音出しを速くしてくれ、演奏者の音程が基準よりも高ければ、全体の音楽が崩れないように、ヴァイオリンの音に合わせて少し高い音を出してくれる。オーケストラの各音を出すプロセスは一つではなく複数あり、それぞれ自律的に動作するが、ヴァイオリンの音を基軸として、お互いに協調しあって動作する。この場合には、ヴァイオリンの音を複数のプロセスが同時に読む必要があるために、単系列のセンサデータを複数のプロセスに対して効率的に提供できるセンサデータベース管理システムが必要になる。

ロボット状況処理

3 番目のアプリケーションは、センサの塊であるロボットである。ロボビーと呼ばれるロボットにはビデオ、マイク、タッチセンサ、超音波センサが付いている [1]。このロボットに人間とのコミュニケーションをさせることを考えたとき、複数人との同時アクセスを行なうには、コミュニケーションプロセスを複数同時に起動し、各コミュニケーションプロセスはセンサデータを読む必要がある。それゆえこの場合には、複数系列のセンサデータを複数のプロセスに提供できるセンサデータベース管理システムが必要になる。

スポーツの技量測定

4 番目のアプリケーションはスポーツの技量測定である。加速度センサを元に、スポーツ選手の技量を測定する研究が近年おこなわれている。センサ数が多いほど技量を正確に測定できるようになるので、複数系列のセンサデータを処理できるセンサデータベース管理システムが必要になる。

2 目的

本プロジェクトの目的は、次の 4 機能をもつ、センサデータベース管理システムを開発することだった。

1) センサデータの鮮度と同期度を高める機能

これを実現するために、近距離のリモートノード上のメモリに対して Write Ahead Logging(WAL) を行なう機能であるメモリ WAL を実装した。メモリ WAL は反応時間とスループットの両面においてディスクを使った WAL よりも優れる。

2) 類似シーケンス検索機能

これを実現するために、ユークリッド距離に基づいて現在のセンサデータと過去の全センサデータを比較する機能を、単系列と複系列の両方について実装した。長さ n である現在のセンサデータを Q とし、過去のセンサデータの一部を S とすると、ユークリッド距離 D は下式により与えられる。

$$D = \sqrt{\sum_{i=1}^{n}(Q_i \quad S_i)^2} \qquad (1)$$

3) リアルタイム検索機能

これを実現するためにシーケンス索引機構を実装した。単系列の場合には Adaptive Piecewise Constant Approximation(APCA)[2] と面積計算による次元削減と R*-tree を組み合わせることにより検索を高速化した。検索は次のように行なわれる。

1: 現在のセンサデータについて APCA 数と面積を計算
2: while (R*-tree から同じものが返される) {
3: 返されたものについて、実データでユークリッド距離を計算
4: }
5: ユークリッド距離が最小のものを答えとする

複系列を検索する場合には、まず単系列で比較をし、そのユークリッド距離が閾値よりも小さければ、他系列のセンサデータを B$^+$-tree 索引構造により取得し、距離計算をおこなう。

R*-tree 索引構造および B$^+$-tree 索引構造は大きな領域を必要とするので、基本的にディスクに置かれるが、高速アクセスのために LRU(Least Recently Used) に従って、その一部はメモリ上に置かれる[1]。

[1] 木構造の探索時間は $O(logn)$ だが、必要な領域は n に対して指数関数的に増加する

4) 拡張性のあるトリガ

これを実現するために、マルチスレッド索引マネージャを実装した。系列ごとに B$^+$-tree と R*-tree を存在させ、それぞれに対してロックを実装した。

3 開発内容

3.1 設計

3.1.1 全体像

開発したセンサデータベース管理システムの全体像を図 1 に示す。

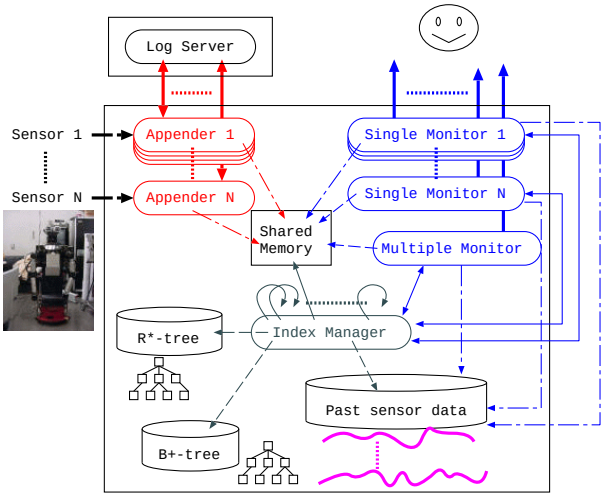

図 1: システム全体像

センサデータベース管理システムは、センサデータを追加する Appender、単系列のセンサデータを監視しつつ、過去の全センサデータの中から類似したシーケンスを探索する single monitor、複系列のセンサデータを監視しつつ、過去の全センサデータの中から類似したシーケンスを探索する multiple monitor、共有メモリ上のデータをローカルディスクに転送すると同時に、時間をキーとして B$^+$-tree を、APCA 数と面積をキーとして R*-tree を作成しつつ、single/multiple monitor の問い合わせに答える索引マネージャからなる。

3.1.2 センサデータの鮮度と同期度を高める機能

センサデータの鮮度と同期度を高めるには、データ挿入速度を高める必要がある。データは読まれる前に永続化されなければならないので、通常はディスクに対してログ先行書き出し (Write Ahead Logging、略して WAL) が実行される。この性能を高めるために、リモートメモリを用いた WAL を実装した。

メモリ WAL のプロトコルは次のようになる。

1: Appender はセンサクライアントからセンサデータを受け取る

2: Appender は Log server にセンサデータを送る
3: Log server はメモリ上にセンサデータを置き、ack を返す
4: Appedner は ack を受け取ると、データをメモリに置く
5: Appender はセンサクライアントに commit を返す

3.1.3 類似シーケンス検索機能

類似シーケンス検索機能は、過去に入った全てのセンサデータと、現在のセンサデータのユークリッド距離を比較する機能である。単系列のセンサデータストリームについて類似検索を実行する single monitor と、複系列のセンサデータストリームについて類似検索を実行する multiple monitor の二種類の機能がある。

センサデータは Appender によって共有領域上に置かれる。このプロトコルは次のようになる。

1: データ用ロックを獲得
2: カウンタ用ロックを獲得
3: データ用メモリを獲得
4: カウンタ用メモリを獲得
5: センサデータをメモリ上に置く
6: カウンタを 1 つ上げる
7: カウンタ用メモリを解放
8: データ用メモリを解放
9: カウンタ用ロックを解放
10: データ用ロックを解放

single monitor は次のプロトコルに従って類似検索を実行する。

1: データ用ロックを獲得
2: データ用メモリを獲得
3: max = データ数 - 窓サイズ
4: for (i = 0; i < max; i ++) {
5: データ [i] から窓サイズ分のシーケンスに対して特徴量を取得 (特徴量は APCA の個数と面積のふたつ)
6: while (索引マネージャから等しい答えが返される) {
7: 返されたものについて実データでユークリッド距離を計算
8: }
9: ユークリッド距離が最小のものを答えとする
10: }
11: センサデータをメモリから窓サイズだけコピー
12: データ用メモリを解放
13: データ用ロックを解放

multiple monitor は次のプロトコルに従って類似検索を実行する。

1: 全データについて、データ用ロックを獲得
2: 全データについて、データ用メモリを獲得
3: max = データ数 - 窓サイズ
4: for (i = 0; i < max; i ++) {
5: あるストリームに対してデータ [i] から窓サイズ分のシーケンスに対して特徴量を取得 (特徴量は APCA の個数と面積のふたつ)
6: while (索引マネージャから等しい答えが返される) {
7: 返されたものについて、実データでユークリッド距離を計算
8: ユークリッド距離が閾値
9: }
10: ユークリッド距離が最小のものを答えとする
11: }
12: センサデータをメモリから窓サイズだけコピー
13: 全データについて、データ用メモリを解放
14: 全データについて、データ用ロックを解放

3.1.4 リアルタイム検索機能

リアルタイム検索を実現するためには、索引を作成する必要がある。センサデータには、次の 2 種類の索引構造が作成される。

1) R -tree

   窓サイズのセンサデータに対して得られた特徴量をキーとし、データの場所を答えとする。

2) B^+-tree

   センサデータの到着時刻をキーとし、データの場所を答えとする。

R*-tree 索引構造および B^+-tree 索引構造は大きな領域を必要とするので基本的にディスクに置かれるが、高速アクセスのために LRU(Least Recently Used) に従って、一部はメモリ上に置かれる。

3.1.5 拡張性のあるトリガ

Monitor の数が増えるにつれて、索引マネージャへの問い合わせ数が増加する。そこで索引マネージャを多重化することで、トリガに拡張性を与えた。

索引マネージャはストリーム毎に R -tree と B+-tree を持つ。それぞれの索引に対して排他制御子を与え、またストリーム毎に処理実体がひとつ与えられる。multiple monitor には専用の処理実体がひとつだけ与えられる。

索引マネージャは appender がメモリ上に置いたセンサデータの中で、すでに single monitor によって読まれたものにつき、B^+-tree 索引と R*-tree 索引を作成し、一括してデータファイルに転送する。また、single monitor と multiple monitor から検索要求があった場合には、索引中のふさわしいデータのデータファイル上でのオフセットを返却する。また、データ転送後にはログサーバに対して、ログ消去メッセージを送る。ログサーバはメッセージを受け取った後に不要なログを消去する。

3.2 実装

実装は RedHat 7.3、Linux Kernel 2.4.19 上で、C 言語を用いておこなった。

3.2.1 センサデータの鮮度と同期度を高める機能

Appender

Appender はプロセスとして実装した。Logger との通信には UDP/IP を用いた。UDP はパケットが落ちる可能性があるので、一定時間経過しても ack が返らない場合には、再送をおこなう実装にした。具体的には alarm システムコールと sigsetjmp システムコールを組み合わせて実装した。

Logger

Logger はプロセスとして実装した。センサ系列をリストでつなぎ、さらに各系列ごとにセンサデータをリストで持たせた。センサデータ追加時にはリストの末尾に新しいデータを加え、抹消時には、抹消時刻までのデータをリストを辿って free させた。

3.2.2 類似シーケンス検索機能

Appender と Monitor の用いるロックにはセマフォをもちいて実装した。すなわち、ロック操作には semget、semctl、semop、ftok などのシステムコールを利用した。メモリは共有メモリをもちいて実装した。すなわち、メモリ操作には shmget と shmat などのシステムコールを利用した。Monitor と索引マネージャの通信には名前付けパイプを使用した。すなわち、fo システムコールを利用した。

3.2.3 リアルタイム検索機能

センサデータおよび索引データの永続化にはファイルを用いた。センサデータの永続化は、構造体を write システムコールと fsync システムコールによってファイルに書き込むことにより行なわれる。ここで、一度に書き込む構造体の数は可変にした。すなわち、もしも複数のセンサデータを書き込んで良い場合には、複数をまとめて書き込む方式をとった。ディスクは一つ書くのも複数まとめて書くのもあまり速度が変わらないという特徴をもつので、このような方式をとった。

3.2.4 拡張性のあるトリガ

索引マネージャの多重化には pthread をもちいた。すなわち、ストリーム毎に与えられる処理実体は、pthread_create システムコールにより与えた。ロックには mutex をもちいた。

索引マネージャにおいて、センサ系列はリストで繋がれる。構造体の中には、ルートノードのファイルオフセット、インデックスファイル名、データファイル名などの情報が収められている。

3.3 評価

3.3.1 評価環境

評価環境として DELL Power Edge 2600 をデータベースシステムのホストとして用いた。このホストのスペックは次の通りである。CPU が Pentium 4 Xeon 2.4GHz 2。メモリは 6GB。ディスクが Ultra 320 SCSI 10000RPM 72GB 4 + 36GB。

ログサーバのホストのスペックは次の通りである。CPU が Pentium II 300MHz である。メモリが 64MB である。ディスクが IBM-DTTA-350640、6GB である。

3.3.2 評価項目

次の 3 項目について評価をおこなう。

1) 実験 1: 挿入時間

   センサデータをデータベースシステムに送ってから commit が返ってくるまでの時間。この時間が短いほど、鮮度と同期度を高められることになる。

2) 実験 2: R*-tree とシーケンシャルスキャンの検索時間

   R*-tree の検索時間とシーケンシャルスキャンの検索時間の比較をおこなう。(x1,y1)(x2,y2) の 2 点からなる矩形を N 個蓄えたあと、あるひとつの矩形を探索するまでに要する時間を測定する。この差が大きいほど、過去のセンサデータからの類似検索が高速になると言える。データは全てメモリ上において比較をおこなった。この理由は、ディスクベースのものでは、データを挿入するのにかかる時間が莫大になってしまうからである。

3) 実験 3: APCA 表現でのシーケンシャルスキャンと、単純なシーケンシャルスキャンの速度比較

   センサデータを APCA 表現にしてからシーケンシャルスキャンをすると、場合によっては元のシーケンスがかなり短くなり、単純なシーケンシャルスキャンよりも探索が高速になる。データは全てメモリ上において比較をおこなった。この理由は、実験 2 同様に、ディスクベースにしては、データを挿入するのにかかる時間が莫大になってしまうからである。

3.3.3 評価結果

1) 実験 1: 挿入時間比較

   表 1 より、メモリ WAL はディスク WAL に比べて数十倍高速だが、同時実行クライアント数が増えるにつれて、メモリ WAL の優位

表 1: 実験 1 の結果

同時実行センサクライアント数	速度比
100	37.6
200	27.6
300	24.3
400	21.7
500	21.6

性は劣化することがわかる。これはログサーバが多重化されていないこと及び、ログサーバホストのスペックがデータベースシステムホストよりもかなり劣ることが原因だと思われる。

2) 実験 2: R*-tree とシーケンシャルスキャンの検索時間比較

表 2: 実験 2 の結果

シーケンス長	速度比
10^2	0.75
10^3	1.8
20^3	2.6
50^3	7.8
10^4	18.0

表 2 より、シーケンス長が 100 の際には、R*-tree よりもシーケンシャルスキャンの方が速くなっているが、シーケンス長が長くなるにつれて、R*-tree はシーケンシャルスキャンよりも高速になっていることがわかる。

3) 実験 3: APCA 表現でのシーケンシャルスキャンと、単純なシーケンシャルスキャンの速度比較

表 3: 実験 3 の結果

シーケンス長		速度比
2	10^3	2
2	10^4	66
2	10^5	622
2	10^6	4912

表 3 に実験 3 の結果を示す。これより、提案手法はデータによってはシーケンシャルスキャンよりもかなり高速になることが示された。この実験において高速化が実現された理由は、データの性質が APCA 表現によるデータ圧縮比率が大きくなる類のデータだったからである。データ圧縮比率がほとんどない類のデータに対しては、検索速度は単純シーケンシャルスキャンのそれとさほど変わらなくなると推測される。

4 開発成果の特徴

海外におけるセンサデータベースシステムの開発には次のようなものがある。

AURORA プロジェクト [3](MIT など)

COUGAR プロジェクト [4](コーネル大学)

STREAM プロジェクト [5](スタンフォード大学)

TELEGRAPH プロジェクト [6](UC バークレー)

NIAGARA CQ プロジェクト [7](ウィスコンシン大学)

これらのシステムと本システムとの違いは、次の 2 点である。

類似シーケンス検索

本システムでは、過去に到着したセンサデータから、現在のセンサデータと類似しているものを検索できるが、上記プロジェクトではそのようなことはできない。

データ永続化

本システムではリモートメモリに対して WAL を行なうことで WAL の高速化を実現しているが、上記プロジェクトでは永続性を無視している。

5 参加企業及び機関

参加企業及び機関はなし。川島英之個人により開発された。

参考文献

[1] 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. 電子情報通信学会研究報告, to appear, April 2003.

[2] 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. データベースワークショップ, 2002.

[3] http://www.cs.brown.edu/research/aurora/.

[4] http://www.cs.cornell.edu/database/cougar/index.htm.

[5] http://www-db.stanford.edu/stream/.

[6] http://telegraph.cs.berkeley.edu/.

[7] http://www.cs.wisc.edu/niagara/.