データベースシステム 109－27
（1996．7．25）

実時間データ獲得システム RTDS の評価

井戸 譲治　島川 博光　竹垣 盛一

三菱電機 産業システム研究所
〒661 尼崎市塚口本町 8-1-1

実世界を対象とするシステムでは，時系列データの獲得・管理機能は必須である．このようなシステムは，データの一元管理という観点からは，一般的なデータベースと機能的に重なる部分が多い．しかしデータ獲得システムは，対象の変化の速度に対応してある時間的制約を守りながら実行されなければならない．一般的なデータベースでは本来このような時間制約についての考慮がないため，これをそのままデータ獲得システムとして用いるのは得策ではない．本稿では，時系列データ処理の性質を活用したアーキテクチャをとる実時間データ獲得システム RTDS について述べ，その性能を評価する．

Evaluation of Real-Time Data Server

Jouji IDO , Hiromitsu SHIMAKAWA and Morikazu TAKEGAKI

Industrial Electronics and Systems Laboratory, Mitsubishi Electric Co.,
8-1-1 Tsukaguchi Honmachi, Amagasaki, Hyougo 661, Japan

Acquisition and managemant of time-sequence data are indispensable to the system interacting real-world. Data acquisition system has time-constraints related to the object of observation, while there are no consideration about such time-constraints in ordinary database system. So, it is not best plan to use ordinary database system as data acquisition system. In this paper, we discuss Real-time Data Server (RTDS) which is the data acquisition system making use of the characteristics of time-sequence data. Also we evaluate the performance of RTDS.

1 はじめに

監視制御システムのような実世界を対象とするシステムでは，外界データの獲得・管理機能は必須である．このようなシステムは，データの一元管理という観点からは，一般的なデータベースと機能的に重なる部分が多い．しかし，観測対象は計算機システムの状況とは無関係に変化してゆくため，データ処理は，対象の変化の速度に対応して，ある時間的制約を守りながら実行されなければならない．一般的なデータベースでは本来このような時間制約についての考慮がないため，これをそのままデータ獲得システムとして用いるのは得策ではない．

このような時間制約を持つデータベースとして，実時間データベースの研究 [5] があるが，不定期に発生するトランザクションのデッドラインを如何にして満たすかに重点が置かれるものが多く，データ獲得システムにはそぐわないように思われる．

また，データ獲得システムが扱うデータは，観測対象の時間的な変化の記録，即ち時系列データであり，このような時間属性を持つデータベースとしては時間的データベースの研究 [4] がある．しかし，データモデルの表現能力に関する研究が多く，時間的データに特有の性質を利用した効率の良い実装についての研究はわずかしかない．

我々は，実時間性を満たすデータ獲得システムの実現のためには，時系列データ処理に特化し，その性質を最大限に利用したシンプルで効率の良い実装法をとることが重要であると考えており，現在このような考えに基づいて現在実時間データ獲得システム RTDS (Real-Time Data Server) の開発を進めている．本稿では，RTDS の性能評価実験の結果を報告する．

2 時系列データ処理

観測対象の状態を示す時系列データの獲得・提供処理を行なうシステムは，特に提供処理面に注目すると，機能的には一般的なデータベースと似通っており，市販の関係データベースなどの流用が考えられる．しかし，時系列データ獲得システムでは，厳格な時間制約が存在する一方で，通常のデータベースシステムにおける複雑な並行制御 / 一貫性管理が必要とされないことが多いため，このような方法は得策とは言えない．むしろ，以下に示すような時系列データ処理に固有の性質を利用して，処理の簡略化を計ることが肝要である．

1. 一定周期あるいは何らかの事象に対応してデータを機械的に取り込む
   人手でデータを入力することは稀であり，制御用ネットワークを通じてセンサなどからのデータを機械的に収集することが要求される．また，時間的に変化する連続値を扱うのが一般的であるため，一定周期での収集が大勢を占める．

2. 一定期間の時系列データを保持
   記憶容量は有限であるため収集された全データを保持することはできないが，通常，ある一定期間のデータを保持しておけば良い．従って，最も古いものから機械的に捨てていけば良い．

3. append only である
   時系列データは観測対象の時間的な変化を記録したものであり，それらに対する変更 / 挿入 / 削除といった処理は通常必要とされない．1., 2. とともに，これらは排他制御が簡略化可能であることを示唆する．

4. 処理のデッドラインが一様でない
   観測対象が時々刻々と変化し，一度取りこぼしたデータは 2 度と取り戻せないために，データ獲得処理には厳密な時間制約が存在する．一方，データ提供処理の場合，例えばトレンドグラフを作成してモニタリングを行なう場合などは，時間制約は比較的緩やかである．また，同じ提供処理をとっても，現在あるいは最近のデータを対象とする場合と，過去のデータを対象とする場合では，後者は時間制約を持たないとして良いことが多い．

5. 一連の場面を単位とする処理がある
   平均値や微積分値など，時間に跨った複数の場面の変数値から導出されるデータが存在する．このとき，入力となる複数の場面データは時間的に散らばったものでなく，ある期間内の一連の場面列である．

6. 検索時に指定される属性が時刻属性に偏る
   保持された時系列データを取り出す場合，何時から何時までという具合に時刻属性についての条件を指定することが多く，他の属性を用いて検索することは稀である．また，関係データベースでは，特定の属性についての条件を満たす組を複数の関係から取り出して連結し新たな関係を作成する，結合 (join)[6] が行なわれるが，時系列データにおいては，異なる周期でサンプリングされたデータを統合したい場合や，ある特定のイベントが起こった時の他の変数の値を知りたい場合，時刻属性についての自然結合あるいはそれに類するもの (厳密な等号でなく幾分かの誤差を許容する場合など) を行なう必要があるものの，時刻属性以外の属性を用いての結合はあまり見られない．

これらはいずれも対象とする時系列データのセマンティクスに依存したものであり，通常の (スナップショット) データベースには見られない性質である．しかし，このような対象依存の性質をうまく利用しなければ，効率の良いデータ獲得システムの実現は困難である．

3 RTDSの概要

前章で述べた時系列データ獲得 / 提供処理についての考察に基づき，我々は実時間データ獲得提供システム RTDS (Real-Time Data Server) を開発している [1][2][3]. 時系列データ処理の性質を活用し，これに特化したアーキテクチャとすることで，処理の簡略化並びに，実時間システムの設計において最も重要な処理時間の見積もりやすさの向上を狙っている．RTDS は次の4種のプロセスから構成される．

獲得プロセス 観測対象のデータを取り込み，必要に応じてディスクにセーブする．データにタイムスタンプを付与し主記憶上のリングバッファに格納する獲得スレッドと，リングバッファ上の時系列データをディスクへセーブする退避スレッドのマルチスレッド構成をとる．また，獲得プロセスは獲得周期毎に設けられる．

提供プロセス 保持された時系列データの検索を行なう．最新データの周期的転送，過去の指定期間のデータの転送など．

総領プロセス クライアントからの要求を提供プロセスに振り分ける．

通信プロセス データの上位ネットワークへの送出並びにクライアントとやりとりするデータの marshaling / unmarshaling を行なう．

RTDS では基本方針として，データの獲得 / 退避及び最新データの周期的な読み出しには周期に基づくデッドラインに応じた比較的高い優先度を与え，過去の時系列の検索処理はこれらの処理のバックグラウンドで行なうようにしている．これは，前章の 4. で述べた性質に基づいている．全てのタスク (具体的にはプロセス / スレッド) に厳密な時間制約を与えるのではなく，適切に階層化することで，実時間性の確保が容易になる．

3.1 データ管理

記憶構造 獲得されたデータは，時刻印を付与された後，主記憶上のリングバッファに獲得順に格納される．リングバッファが一巡した後は，新たなデータの獲得の際には最古のレコードが上書きされる．また，一定数の獲得が行なわれる毎に，リングバッファ上の時系列データはディスクへセーブされる．時系列データを保持する連続ファイルもリングとして管理され，古いものから上書きされてゆく．

リングバッファのサイズ及び退避のトリガとなる獲得回数を適切に設定することで，獲得スレッドと退避スレッドをそれぞれの時間制約を守るように動作させ，かつリングバッファ自身をロックすることなく，獲得されたデータを書き潰す前にディスクにセーブすることができる．

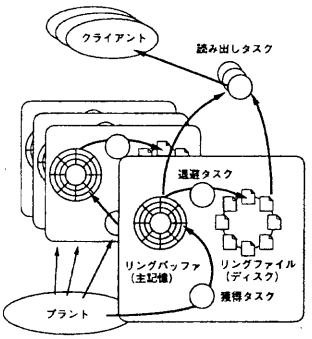

図 1: データ管理モデル

また，獲得された時系列データに対する削除 / 修正を認めていないため，リングバッファ及びファイル上で，時系列データは時刻順に並ぶことが保証される．このような記憶構造により，

- 最近のデータは主記憶上にあるため高速アクセスが可能
- そのままバイナリサーチによる検索が可能
- 時刻属性に基づく結合 (join) 操作が比較的効率良く実現できる
- 平均値や微積分値といった，一定期間の一連の場面列へのアクセスが容易

といった利点が生まれる．

排他制御 リングバッファ及びリングファイルは，獲得プロセス毎に用意される．これらからの読み出しが複数の提供プロセスにより行なわれるのに対し，リングバッファへの書き込みは対応する獲得スレッドにより，またリングファイルへの書き込みは対応する退避スレッドにより行なわれる．つまり複数の読み出しに対して書き込みは単一である．また，書き込みを行なうスレッドは，常に最古のレコードを上書きしてゆく．従って上書きをされそうなレコードの読み出しさえ注意すれば十分である．

そこで，リング構造の最新 / 最古位置を保持した管理情報へのアクセスのみを優先度継承セマフォを用いて排他制御し，リングバッファ及びリングファイル上のデータそのものへはロックせずにアクセス可能とした．万一読み出し途中のデータを書き潰された場合のために，楽観的読み出しを行なう．即ち，最初に時刻印を読み，各変数を読んだ後，最後にもう一度時刻印を読み，最初に読んだ時刻印と最後に読んだ時刻印の値が一致していなければ，そのデータを読みだす過程で獲得スレッドに上

書きされたと判断して，このデータを無効として破棄し，再度ファイルから読み出すことで補償する．

ロックの対象となる管理情報は小さなものであり，読み書きに必要な時間は無視できるほど小さい．また，優先度継承セマフォを用いているため，優先度逆転による不当なブロックは起こらない．従って共有データへのアクセスで生じるはずのブロック時間は無視できるほど小さくなる．そのため，各スレッドの最悪実行時間が見積もれ，静的なスケジューリング判定が可能となる．

3.2 時系列データの提供

前章の5,6で述べたように，時系列データの提供では，時刻属性を中心に考えることで多くの処理の効率化が計れる．RTDSでは簡略化のため，時系列の期間とその場面間の時間的間隔(以後レートと呼ぶ)，許容誤差および取り出すべき変数のみを検索パラメータとして指定可能とした．

時系列の検索 前節で述べたように，リングバッファ及びリングファイル上では各場面データは時間順に並んでいる．従って，時刻印についての検索はバイナリサーチを用いて効率良く行なうことができる．また，ファイル上に存在する場面データの検索を効率良く行なうために，各ファイルが保有するデータの期間を示すファイルインデックスを設け，インデックス上のサーチでファイルを特定した後，ファイルを主記憶上のキャッシュにロードし，キャッシュ内をバイナリサーチすることにした．ファイルインデックス自身もリングバッファなどと同様にリング状に管理されるため，更新は容易である．時系列データの検索は通常は単一場面でなく一定期間の一連の場面列として要求されるため，指定された期間の先頭場面をバイナリサーチで求め，以降の場面はレコードを読み進めることで求めることにした．この際，i番目の場面は，期間開始時刻をb，検索レートをr，許容誤差をaとして，区間$[b+ir, b+ir+a)$に含まれる場面のうち時刻印の最も小さいものと規定する．

時刻印に基づく結合 前節で述べたように時系列データは獲得周期毎に記憶・管理されるため，獲得周期の異なるデータを統合して見るために時刻印に基づく結合を行なう．RTDSでは，時刻印が一致する場合のみ結合するraw-joinと，i番目の場面として検索された場面データを時刻印の一致/不一致に関わらず結合し，結合して得られた場面の時刻印を新たに$b+ir$とするaligned-joinをサポートする．

一般的な結合では，関係内の組が結合条件に用いる属性の値で整列されていないと，最悪の場合これらの関係の直積の大きさに相当する回数の結合条件のチェックが必要となるため，非常に効率が悪い．しかし，RTDSでは場面データが獲得された時点ですでに時刻順に整列されているため，上述のような時刻印に基づく結合の場合，一旦開始場面を求めてしまえば，あとは順に読み進めつつ時刻印を比較すれば良い．

4 評価

RTDSを試作し，提供処理の性能について評価を行なった．評価システムの構成及び検索要求を以下に示す．

H/W CPU:Pentium90MHz, RAM:32MB, HD:1GB

OS LynxOS Ver2.3

S/W構成 クライアントマシンは接続せず，検索要求を記述したファイルを通信プロセス内の受信スレッドに読み込ませた．また，疑似データを生成し獲得プロセスとの共有メモリに書き込むデータジェネレータプロセスを設け，周期50msecでデータを更新させた．獲得プロセスは2つ設け，それぞれ64bit 浮動小数点型データ及び32bit 整数型データからなる1024byteのデータを獲得させた．以後前者を獲得プロセスA，後者を獲得プロセスBと呼ぶことにする．獲得プロセスA，Bともに，リングバッファは最大1000場面を，リングファイルは最大15000場面を保持できる大きさとし，獲得100回毎に退避を行なわせた．また，獲得プロセスBの獲得周期を獲得プロセスAの獲得周期の2倍とした．

検索要求 検索スキーマとしては，獲得プロセスAが収集するデータから16変数を抜き出したもの(以降のグラフ内ではsingle schema)と，獲得プロセスA，Bが収集するデータからそれぞれ16変数を取り出して結合したもの(同join 2 schema)の2つを設定した．獲得プロセスAの獲得周期を検索レートとし，要求を発した時刻を基準時刻として過去へ指定数の場面を読み出させ，要求送出から全場面が通信プロセスに到着し終えるまでの時間を求めた．このとき場面数を100から順に増やして行き，これを1セットとして5セットを行ない平均を求めた．

4.1 結合処理の性能

獲得プロセスA，Bの獲得周期をそれぞれ100msec, 200msecとし，提供プロセス内のキャッシュサイズを512KB(場面数500)として，結合を行なう場合(join 2 schema)と行なわない場合(single schema)を比較した．

結合を行なう場合で場面数1500のときに処理時間が短くなっているのは，前回の検索(1000場面)時にキャッシュに読み上げたデータがヒットしたためと考えられる．

さて，同じ大きさの場面データを，獲得プロセスAが獲得プロセスBの倍の周期で獲得を行なうことから，同一期間の時系列データを保持するのに必要な記憶容量を比較すると，AはBの倍である．従って，同一期間

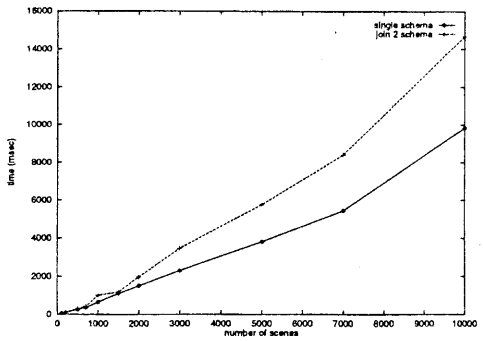

図 2: 結合処理の有無の比較

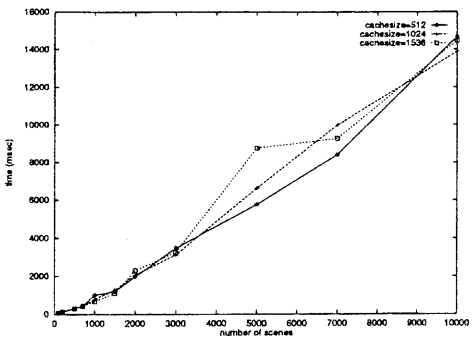

図 4: キャッシュサイズを変化させた場合の比較 (join あり)

のデータを読み出す場合，A，B，及び両方のリングファイルからの読み出し量の比は 2 : 1 : 3 となる．これは実験結果の特に 3000 場面以降でよく一致している．

一方場面数が数百程度までは結合を行なう場合と行なわない場合とでの処理時間はほぼ一致しており，1000 場面以降で差が開き出している．これはリングバッファサイズが 1000 場面であることに関係している．

このことから，検索時間はディスクからの読み出し量に大きく依存しており，時刻印に基づく結合処理は結合を行なわない処理と大差ない程度にまで効率良く実行されていると言える．

4.2 キャッシュサイズ

上記の実験と同様の獲得周期で，提供プロセス内のキャッシュサイズを 512KB(場面数 500)，1024KB(同 1000)，1536KB(同 1500) と変化させ，比較した．

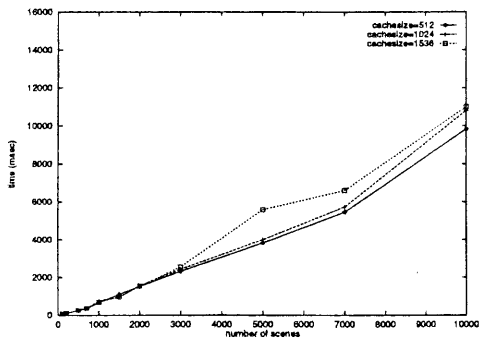

図 3: キャッシュサイズを変化させた場合の比較 (join なし)

RTDS では連続ファイルを用いているため，複数のファイルを読む際，ひとつのファイルを読み終えてから次のファイルに移る場合と，複数のファイルを渡り歩きながら少しずつ読む場合では，シーク量の差により読み出し時間に大きな開きが出る．上記のようにひとつのファイルを読み終えてから次のファイルへ移るのが理想ではあるが，仮に複数ファイル間を渡りながら読むとしても，一回の読み込み量が大きければ，それだけシーク量は小さくなるはずである．

結果は，キャッシュサイズを増大しても処理速度の大きな向上は認められなかった．このことから，今回の実験での最小のキャッシュサイズ 512KB(場面数 500) が，シーク量を無視できる程度にまで押えるのに十分なサイズであったと考えられる[1]．

逆にキャッシュサイズ 1536KB(場面数 1500) の場合，要求期間に当てはまらない不要なデータの読み出しの影響により，場面数によっては処理速度が低下した．

次に，キャッシュサイズを 512KB(場面数 500) とし，要求場面数を細かく変化させて，検索処理の特性を調べた．

要求場面数 100 ～ 900 では，要求期間の全データがリングバッファ上に存在するため，場面数に対してほぼリニアに検索時間が増加している．リングバッファ上に存在する期間をわずかに越える期間を読み出そうとするとディスクから読み出す無効データの量が大きくなるために処理時間が急激に劣化するはずであるが，実験結果においても場面数 1000 でそれが現れている．

また，場面数 1100 で処理時間が小さくなっているのは，その前に行なわれた場面数 1000 の検索でキャッシュされたデータがヒットしたためであろう．続く場面数 1200 及び 1300 の検索でも同様のことが言える．さらにキャッシュヒットしたときの処理時間が要求場面数 100 ～ 900 の部分のグラフの延長上に存在することから，ディスク

[1] 以前の版の RTDS では，キャッシュを用いずに，インデックス上で検索された場面毎にディスクから読み出していたため，結合処理は結合を行なわない検索の数十倍の時間を要した

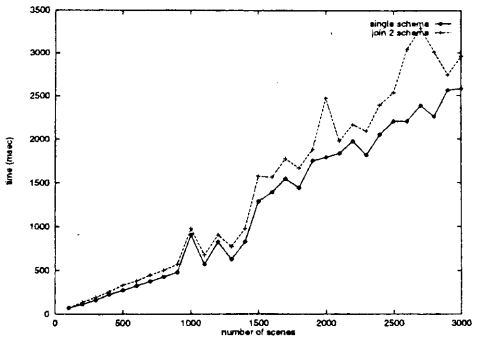

図 5: キャッシュを用いた検索時間の特性

アクセスを伴わない主記憶上での検索処理時間は結合処理を行なうか否かに関わらずほぼ場面数に比例し，かつディスクアクセスを伴う場合はそれに要する時間が支配的になると言える．より小さい単位でのキャッシングを行ない，無効場面の読み出しを抑制する必要があろう．

4.3 獲得プロセスの負荷との関連

キャッシュサイズを 512KB(場面数 500) とし，獲得プロセス A，B の獲得周期をそれぞれ (50msec,100msec), (100msec,200msec) と変化させて，これらの場合での検索時間を比較した．

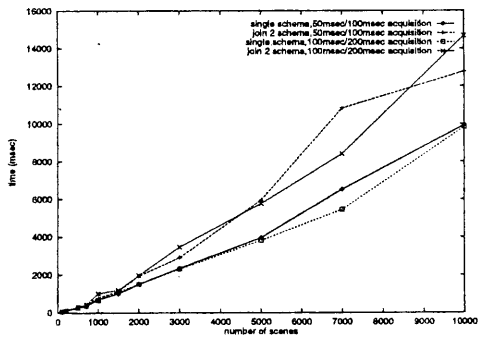

図 6: 獲得周期を変化させた場合の比較

(50msec,100msec) の場合に要求場面数 7000 で処理時間が長くなっているのは，獲得周期が短くなって検索処理に与えられる CPU 時間が短くなったというよりもむしろ，ちょうど無効場面のキャッシュ量が大きくなる場面数だったためと思われる．要求場面数 10000 において 2 つの場合の処理時間が逆転していることからも，そのように考えるのが妥当であろう．従って 2 つの場合で検索時間に大きな変化は無いと言える．

5 おわりに

本稿では，実時間データ獲得システム RTDS の性能評価実験を行なった．実験の結果，データ検索 / 提供処理がデータ獲得処理の実時間性を阻害することなく現実的な処理速度を確保しているものの，ディスクアクセスに関しては改善の余地があることが判明した．より細やかな単位でのキャッシングによる無効場面の読み込みの削減，獲得プロセスと同様のディスクアクセス部とデータ処理部のマルチスレッド化などが今後の課題である．

参考文献

[1] H.Shimakawa, H.Ohnishi, I.Mizunuma, M.Takegaki: Acquisition of Temporal Data for Real-Time Plant Monitoring, *Proc. of RTSS'93* (1993)

[2] 島川 博光，井戸 譲治，浅野 義智，竹垣 盛一: 実時間データサーバへの ECA 機能の導入，電子情報通信学会研究報告 CPSY94 - 116 ～ 124 , pp.49–56 (1995)

[3] 井戸 譲治，島川 博光，浅野 義智，竹垣 盛一: 周期・非周期データを統合する実時間データサーバ，電子情報通信学会研究報告 DE95 - 38 ～ 49 , pp.63–70 (1995)

[4] A.Tansel, et al.: *Temporal Databases - Theory, Design, and Inplementation*, The Benjamin / Cummings Publishing Company (1993)

[5] K.Ramamritham: Real-Time Databases, *International Jounal of Distributed and Parallel Database*, Vol.1, No.2 (1993)

[6] C.J.Date: *An Introduction to Database Systems*, 5th Edition, Addison Wesley (1990)