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

川島英之[†] 遠山元道[††]
今井倫太[††,†††] 安西祐一郎[††]

本論文は，連続的なセンサデータストリームをつねに監視する必要があるアプリケーションに対して，優れた鮮度と永続性を保ったセンサデータを提供する手法を示す．データの鮮度と永続性を保つためには，永続化にともなう鮮度劣化を防ぐ必要がある．本論文は，この問題を解決するため，処理負荷の少ない WAL 方式を提案する．提案方式は，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読まれないセンサデータに対して投機的な WAL を適用することで，処理負荷を減らす．本論文では，提案プロトコルを実装し，提案プロトコルと TCP のみ用いるプロトコルについて，比較実験を行った．その結果，同時実行ストリーム数が多い場合に，提案プロトコルは，比較手法より優れた鮮度を持つデータを提供できた．

Providing Persistence for Sensor Data Streams Using Remote Memories

Hideyuki Kawashima,[†] Motomichi Toyama,[††] Michita Imai[††] and Yuichiro Anzai[††]

This paper proposes a new WAL protocol that provides sensor data with both good freshness and persistence for applications that continuously monitors sensor data streams. To keep freshness and persistence of data, degradation of freshness with persisting must be avoided. To solve this problem, this paper applies WAL to memories on remote hosts and also applies speculative WAL for sensor data unread by applications. This paper implements the new WAL protocol and evaluates the performance compared with a protocol that uses only TCP. The result of the experiments showed that the new WAL protocol provided sensor data with better freshness than the protocol that uses only TCP.

1. はじめに

本論文の目的は，センサデータストリームを連続的に監視するアプリケーションに対して，優れた鮮度と永続性を持つデータを提供する手法を示すことである．人間とコミュニケーションするためのセンサネットワーク[1)]やロボット[2)]は，人間と豊かなコミュニケーションを実現するために，途切れなくシステムに到着するセンサデータストリームから人間の状況を実時間認識し，かつ認識された状況と類似する過去の状況を実時間検索する必要がある．状況の実時間認識に用いるセンサデータには優れた鮮度が必要であり，過去の類似状況を検索するにはセンサデータを永続化しなければならない．したがって，これらのアプリケーションに対して，データベースシステムは優れた鮮度と永続性を持つセンサデータを提供する必要がある．

永続化処理は時間を要するから，データ鮮度を劣化させる原因となる．それゆえ，データに優れた鮮度を与えるには，永続化に要する処理負荷を減らし，処理を高速化する必要がある．

データ永続化処理を高速化するには，2つの従来手法がある．1つは，リモートホストのメモリ（主記憶）を永続化デバイスとすることで処理負荷を減らし，WAL を高速化する手法である．もう1つは，WAL 処理を間引いて負荷を減らすことで，処理を高速化する手法である．WAL 処理がされないデータは障害時に失われるため，それらは補間により復旧される．

† 慶應義塾大学大学院理工学研究科開放環境科学専攻
Science for OPEN and Environmental Systems, Graduate School of Science and Technology, Keio University
†† 慶應義塾大学理工学部情報工学科
Department of Information and Computer Science, Faculty of Science and Technology, Keio University
††† 科学技術振興事業団さきがけ研究 21
Precursory Research for Embryonic Science and Techonology, Japan Science and Technology Corporation

リモートホストのメモリを永続化デバイスとする手法の問題は，全データに永続性を保証しようとするために，WAL 処理の負荷が重くなるか，もしくは WAL 負荷を軽減するためにすべての WAL が投機実行され，データが永続化されない可能性が生じることである．一方，WAL を間引く手法の問題は，WAL 処理の負荷を軽減するために，WAL 処理を間引くデータに永続性が与えられないことである．

そこで本論文では，リモートホストのメモリを永続化デバイスとしながら，さらに，アプリケーションが読まないデータには投機的な WAL を適用する一方，アプリケーションが読むデータには永続性を保証する WAL を適用する手法を提案する．アプリケーションに読まれるデータには優れた鮮度と永続性を提供し，アプリケーションに読まれないデータには高確率で永続性を与えることで，提案手法は目的を遂げる．

本論文の構成は以下のとおりである．2 章では本研究の必要性，用語定義，そして関連研究を述べる．3 章では優れた鮮度と永続性を持つデータを提供する手法を提案する．4 章では提案手法の設計および実装を述べる．5 章では実験により提案プロトコルを評価する．6 章では専用プライベートネットワークにおけるマルチキャストの信頼性と提案手法の限界点について議論し，最後に 7 章で結論を述べる．

2. 前　　提

2.1 センサデータストリーム永続化の必要性

2.1.1 従来研究の姿勢

センサデータストリームを扱うデータベースシステムを開発するプロジェクトには，COUGAR[3]，AURORA[4]，TELEGRAPH[5]，STREAM[6]等がある．このうち，AURORA プロジェクト[4]では，次の 3 つのアプリケーションがあげられている．(1) 兵士のセンサデータ（血圧，心拍数，位置）を監視する，軍事アプリケーション．(2) 複数の株式取引所から報告される株データストリームを監視する，金融解析アプリケーション．(3) 大量のオブジェクトの位置を監視する，追跡アプリケーション．

これらのアプリケーションがデータベースシステムに要求するセンサデータの性質は，優れた鮮度である．なぜなら，センサデータの鮮度が低ければアプリケーションは過去のデータを読むことになるからである．

上記プロジェクトでは，センサデータの永続化が研究されていない．この理由は，上記アプリケーションが過去のセンサデータストリームを使わないからである．したがって，戦場における兵士の身体状況（血圧，心拍数，位置）の変化を保存しておき，それらを後で解析することで，兵士の訓練方法や装備を改良するといったアプリケーションは上記プロジェクトの対象外である．

2.1.2 本論文の姿勢

一方，本論文が対象とするアプリケーションは過去のセンサデータストリームを使う．その一例として，Robovie[2]のようなコミュニケーションロボットの状況認識処理がある．この処理では，人間と豊かなコミュニケーションを実現するために，センサデータを使って人間の状況を認識するだけでなく，認識された状況と類似する状況を過去のセンサデータストリームから探索することも求められる．

したがって，本論文のアプリケーションがデータベースシステムに要求するセンサデータの性質は，優れた鮮度と永続性の両方である．センサデータの鮮度が不十分ならば，コミュニケーションロボットは過去のセンサデータを使って状況を認識し，過去の状況に適した行動をとるから，人間はコミュニケーションロボットの反応の遅さに不快感を覚える．センサデータに永続性がなければ，障害時に失われたセンサデータに表される状況を，将来において検索することが困難になる．

ここで注意すべき点は，永続性のないセンサデータをアプリケーションに読ませるべきではないことである．データベースシステムとは，あるコンピュータシステムにおいてデータを集中的に管理することで，データの一貫性を保証するソフトウェアである[7]．したがって，ある時点では存在するが将来において消えうるような，存在性を保証できないデータを，データベースシステムはアプリケーションに提供するべきではない．

そして本論文で対象とするアプリケーションが永続性のないデータを読むと次のような問題が発生する．それは永続性のないデータをコミュニケーションロボット内の状況認識処理アプリケーションが読んでしまった場合，人間とコミュニケーションロボットの間で作られたコミュニケーションの流れが狂うことである．たとえば，コミュニケーションロボットと人間が喧嘩していたが，仲直りをして，その直後に電源障害が発生したとする．センサデータが永続化されていれば障害直前のセンサデータがデータベースにあるから，状況認識処理は復旧時にセンサデータから人間とのコミュニケーションの流れを生成することができ，コミュニケーションロボットは人間とのコミュニケーションを円滑に再開できる．しかしセンサデータが永続化され

ていなければその流れを生成できないから，復旧後にコミュニケーションロボットは人間に対して嫌悪感を持つという状況が発生してしまう．このように流れの一部が抜け落ちることは，人間とコミュニケーションをすることを目的としたコミュニケーションロボットに起きるべきではない．この事態を避けるために，コミュニケーションロボットはセンサデータを永続化する必要がある．

2.2 用語定義

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センサモニタが読める s を s_{read}，センサモニタが読めない s を s_{unread} と表記する． □

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． □

定義 2.3（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電源障害によって停止しても，データベースシステムに復旧できる s の性質」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，確率的に保証される p を p_{weak} と表記する．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． □

定義 2.4（鮮度）

本論文では，アプリケーションが読み出した s_{read} の鮮度を $\mathsf{f}(\mathsf{rd}(s_{read}))$ と表記し，$\mathsf{f}(\mathsf{rd}(s_{read}))$ を次のように定義する．

$$\mathsf{f}(\mathsf{rd}(s_{read})) = \mathsf{rt}(s_{read}) - \mathsf{at}(s_{read})$$

$\mathsf{rd}(s_{read})$ はセンサモニタが s_{read} を読み出す（ReaD）演算を表し，$\mathsf{rt}(s_{read})$ は $\mathsf{rd}(s_{read})$ を行った時刻（Read Time）を表す． □

$\mathsf{rt}(s_{read}) \geq \mathsf{at}(s_{read})$ だから，$\mathsf{f}(\mathsf{rd}(s_{read})) \geq 0$ である．$\mathsf{f}(\mathsf{rd}(s_{read}))$ は小さいほど優れる．

定義 2.5（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． □

2.3 関連研究

ここでは関連研究として，$\mathsf{p}(s)$ の従来手法と，$\mathsf{p}(s)$ を間引くことで $\mathsf{f}(\mathsf{rd}(s_{read}))$ を下げる手法を述べる．

2.3.1 $\mathsf{p}(s)$ の従来手法

ここでは，$\mathsf{p}(s)$ の従来方法として，D-WAL，N-WAL，L-WAL，そして T-WAL について説明する．

D-WAL プロトコルはディスクに対して WAL を行うプロトコルであり，$\mathsf{p}(\forall s) \rightarrow p_{strong}$ である．PostgreSQL[8] 等の多くのデータベースシステムがこのプロトコルを採用しているが，文献 9) に示されているように，頻繁にシステムに到着する s を処理しようとすると，ディスクの機械的性質のためにパフォーマンスが急激に劣化する．それゆえ，D-WAL は本論文で対象とするアプリケーションには適用すべきではない．

N-WAL プロトコル[10]は，2 相コミットプロトコルに則り，2 台以上のリモートメモリに WAL を実行するプロトコルであり，$\mathsf{p}(\forall s) \rightarrow p_{strong}$ である．N-WAL はディスクを使用しないから D-WAL より高速だが，2 相コミットプロトコルの負荷が高い．それゆえ，その処理負荷を軽減すべきである．

L-WAL プロトコル[11]は専用プライベートネットワーク上で，複数台のログサーバにログパケットをマルチキャスト送信し，ログサーバからの ack を受信しないことで，$\mathsf{p}(s)$ を N-WAL よりも高速化したプロトコルである．ただし，ログサーバから ack を受信しないから $\mathsf{p}(\forall s) \rightarrow p_{weak}$ であり，N-WAL を本論文の対象アプリケーションに適用できない．

T-WAL プロトコル[12]は L-WAL の高速案である．アプリケーションの要求する時間的一貫性が，s の到着周期に比べて大きい場合に，時間的一貫性が守られる範囲で複数個の s のログをまとめてログサーバに送信することで，$\mathsf{p}(s)$ を高速化する．T-WAL は L-WAL 同様，$\mathsf{p}(\forall s) \rightarrow p_{weak}$ だから，T-WAL を本論文の対象アプリケーションに適用できない．

2.3.2 $\mathsf{p}(s)$ を間引き $\mathsf{f}(\mathsf{rd}(s_{read}))$ を下げる手法

$\mathsf{p}(s)$ を実行しなければマシンの負荷が減るから，$\mathsf{f}(\mathsf{rd}(s_{read}))$ は下がる．障害時には，$\mathsf{p}(s)$ を実行しなかった s は失われるが，補間で $\mathsf{v}(s)$ をある程度復旧できる可能性がある．しかし，この手法を本論文のアプリケーションへ適用することは好ましくない．なぜなら $\mathsf{v}(s)$ を多数補間することにより，類似状況の探索が困難になるからである．この理由を以下に述べる．

コミュニケーションロボット Robovie に装着されて

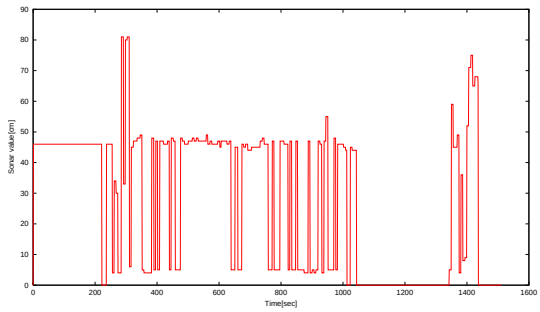

図 1 Robovie の超音波センサデータ値 v(s)
Fig. 1 Ultrasonic sensor data v(s) at Robovie.

いる超音波センサから得られる v(s) は**図 1** のように急激に変化する．そのため，補間値と真値の差は大きくなりうる．

そして，類似検索の類似尺度がユークリッド距離であり，1 次元の類似検索を行うとする．n 点から成る S_A と S_B のユークリッド距離 $Dist_{euclid}(S_A, S_B)$ は次式のように計算される．

$$Dist_{euclid}(S_A, S_B) = \sqrt{\sum_{i=1}^{n}(S_{A_i} - S_{B_i})^2}$$

S_A と S_B に補間された値があるほど，$Dist_{euclid}(S_A, S_B)$ は真値からずれるから，類似状況の探索精度が劣化する．類似尺度がダイナミックタイムワーピング距離[13]であれば，劣化度合はユークリッド距離よりも緩やかだが，劣化は避けられない．補間すべきデータ点数が増えるほど，類似状況の探索精度は劣化する．この手法を用いると，間引かれるデータは障害時に必ず失われるから，探索精度は必ず劣化する．望まれるのは，補間すべきデータ点数を低負荷で減らす手法である．

2.4 問題の定式化

本論文で解くべき問題は次のように定式化される．「p(s_{read}) → p_{strong} を満たしながら f(rd(s_{read})) を下げること」

3. 提　　案

本論文では，従来手法を組わ合わせることにより，p(s_{read}) → p_{strong} を満たしながら f(rd(s_{read})) を下げる手法を提案する．提案手法は，p(s_{read}) → p_{strong} と p(s_{unread}) → p_{weak} を同時に満たすことで WAL の処理負荷を減らし，f(rd(s_{read})) を下げる．

提案手法は，p(s_{read}) → 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 $\mathrm{p}_{\mathrm{tcp}}(s_{read})$ と表記する．$\mathrm{p}_{\mathrm{tcp}}(s_{read}) \rightarrow p_{strong}$ である．また，提案手法は p(s_{unread}) → 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 p(s_{unread}) → 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，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 $\mathrm{p}_{\mathrm{udp}}(s_{unread})$ と表記する．$\mathrm{p}_{\mathrm{udp}}(s_{unread}) \rightarrow p_{weak}$ である．

提案手法の特徴は，s_{unread} に $\mathrm{p}_{\mathrm{udp}}(s_{unread})$ 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．$\mathrm{p}_{\mathrm{udp}}(s_{unread})$ に必要な負荷は $\mathrm{p}_{\mathrm{tcp}}(s_{read})$ 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 $\mathrm{p}_{\mathrm{udp}}(s_{unread}) \rightarrow p_{weak}$ 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，$\mathrm{p}_{\mathrm{udp}}(s_{unread})$ 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 $\mathrm{p}_{\mathrm{udp}}(s_{unread}) \rightarrow p_{weak}$ である $\mathrm{p}_{\mathrm{udp}}(s_{unread})$ 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．それゆえ提案手法を用いれば，将来における過去のコミュニケーション履歴解析時におけるパターンマッチングにおいて，ユークリッド距離とダイナミックタイムワーピング距離のいずれを用いたとしても，p(s) を間引くだけの場合よりも，補間すべきデータ点数を少なくできる．それゆえ提案手法は p(s) を間引くだけの手法よりも高精度のパターンマッチングを実現する．

4. 設計と実装

本章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 2** に示されるシステム上で動作する．図 2 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

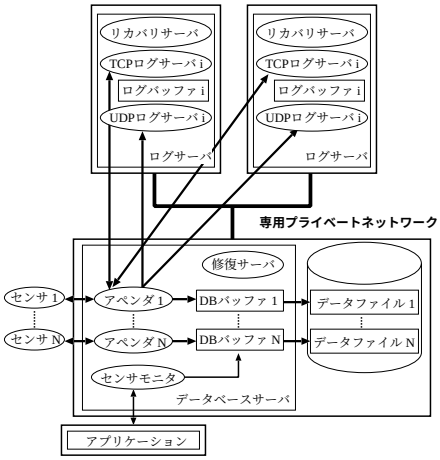

図 2 システム構成
Fig. 2 System configuration.

表 1 ログパケットの構造
Table 1 Structure of log packet.

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

表 2 DB バッファの構造
Table 2 Structure of DB buffer.

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

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．たとえば，図 2 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 1 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 1 つのみ存在する．

4.1 データベースサーバ

4.1.1 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 1** に示す．表 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 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．

4.1.2 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 3** に示し，図 3 中の流れに沿って，アペンダスレッドのプロトコルを説明する．

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．たとえば 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，2 つに 1 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

```
 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 3 アペンダスレッドのプロトコル
Fig. 3 Protocol of appender thread.

切替えを行い，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 項で述べる．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）．

4.1.3 ディスク転送スレッドのプロトコル

図 3 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 4** に示す．図 4 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 3 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．

4.2 センサモニタのプロトコル

センサモニタは周期的に $\mathsf{rd}(s_{read})$ を実行する．このプロトコルを**図 5** 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

```
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 4 ディスク転送スレッドのプロトコル
Fig. 4 Protocol of disk transfer thread.

```
 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) -> p_strong){
 5:     sbc 番の s_read を取得;
 6:     return;
 7:   }
 8:   else sbc --;
 9: }
10: データバッファ切替え;
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; return;
```

図 5 センサモニタのプロトコル
Fig. 5 Protocol of sensor monitor.

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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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 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 10 ms 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 に示す手法を用いる．

4.3 ログサーバ

4.3.1 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 3** に示す．ログバッファ i は UDP ログサーバ

表 3 ログバッファの構造
Table 3 Structure of log buffer.

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 フラグまたは UDP フラグ）

```
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 6 UDP ログサーバスレッドのプロトコル
Fig. 6 Protocol of UDP log server thread.

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 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．

4.3.2 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 6** 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．

4.3.3 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 7** 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．

4.3.4 修復プロトコル

ここでは，図 7 中の 10〜12 行における修復プロトコルについて述べる．

図 7 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 番目から前方へたどって，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．

このとき，**図 8** 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

```
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:    データバッファ切替え;
16:    break;
17:  }
18:  ack を送信;
19: }
```

図 7 TCP ログサーバスレッドのプロトコル
Fig. 7 Protocol of TCP log server thread.

```
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 8 修復プロトコル
Fig. 8 Repair protocol.

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．それゆえログ抜けが多発する環境では，修復作業はパフォーマンスを劣化させる原因となる．

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を行わない．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．前述のとおり，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

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 at(s')>at(s) の 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 9 リカバリプロトコル
Fig. 9 Recovery protocol.

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してしまう可能性があるから，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．

4.4 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，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 9** に示す．

5. 評　　価

本章では，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 $\mathsf{rd}(s_{read})$ を実行するセンサモニタ（図 2 参照）が得る $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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．

5.1 実験条件

センサモニタはある 1 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 秒と設定される．そしてセンサモニタの 100 回の測定を持って 1 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 10 ms 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 1,024 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．

5.2 実験内容

次の 3 方式について，$\mathsf{f}(\mathsf{rd}(s_{read}))$ 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 $\mathsf{p}_{\mathsf{tcp}}(s_{read})$ を実行し，s_{unread} には $\mathsf{p}_{\mathsf{udp}}(s_{unread})$ を実行する．

(2) TCP 方式
TCP 方式は，s_{read} には $\mathsf{p}_{\mathsf{tcp}}(s_{read})$ を実行し，s_{unread} には $\mathsf{p}(s_{unread})$ を実行しない．

(3) D-WAL 方式
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．

5.3 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 4** に示す．ネットワークには 100 M イーサネットを使用した．

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** に示す．

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．

表 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
Table 4 Specifications of database server and virtual sensor data generation host.

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 2.4 GHz シングル
ディスク	Ultra ATA 100, 7,200 rpm
メモリ	1 GB
ネットワークインタフェース	100 Mb Fast Ethernet

表 5 ログサーバ用ホストの仕様
Table 5 Specifications of log server hosts.

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 300 MHz シングル
ディスク	IBM-DTTA-350640
メモリ	64 MB
ネットワークインタフェース	100 Mb Fast Ethernet

5.4 実験結果

5.4.1 平均鮮度に関する結果

センサモニタが得た $\mathsf{f}(\mathsf{rd}(s_{read}))$ の平均値を，提案方式について**図 10** に，TCP 方式について**図 11** に，そして D-WAL 方式について**図 12** に示す．図 10 と図 12 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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ に比べて，提案方式の間引き率 0%の $\mathsf{f}(\mathsf{rd}(s_{read}))$ は 61 倍程度小さく，提案方式の間引き率 100%の $\mathsf{f}(\mathsf{rd}(s_{read}))$ は 85 倍程度小さい．そして図 10 と図 11 から次のことが分かる．

- 提案方式は本論文の問題を解決できたこと
  図 10 における間引き率 100%のグラフと，図 11 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分かる．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，すべての 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 1 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 $\mathsf{f}(\mathsf{rd}(s))$ 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

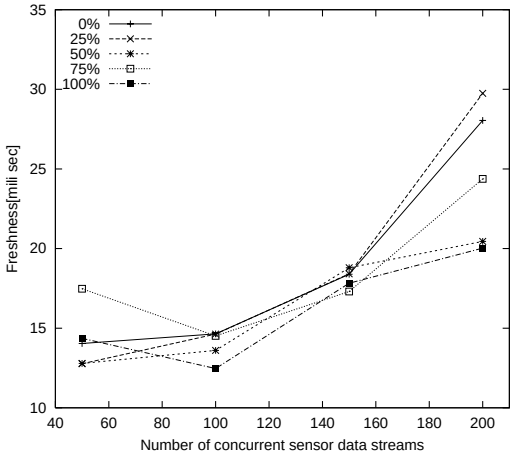

図 10 提案方式（$\mathsf{f}(\mathsf{rd}(s))$ の平均）
Fig. 10 Proposed method (average of $\mathsf{f}(\mathsf{rd}(s))$).

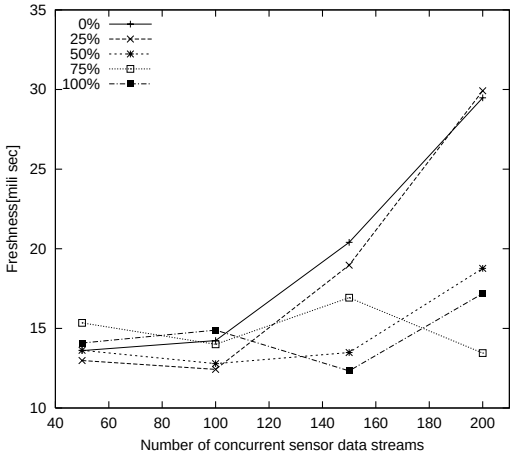

図 11 TCP 方式（$\mathsf{f}(\mathsf{rd}(s))$ の平均）
Fig. 11 TCP method (average of $\mathsf{f}(\mathsf{rd}(s))$).

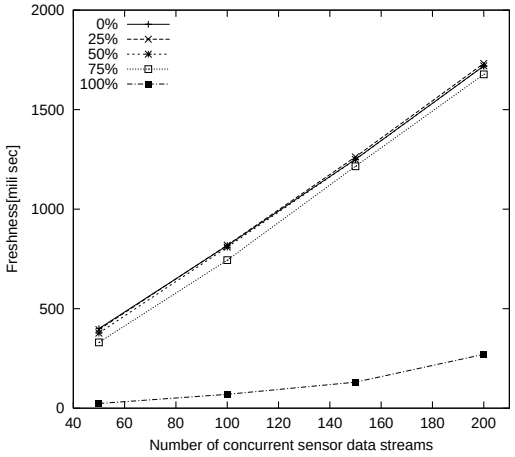

図 12 D-WAL 方式（$\mathsf{f}(\mathsf{rd}(s))$ の平均）
Fig. 12 D-WAL method (average of $\mathsf{f}(\mathsf{rd}(s))$).

間引き率 0%に比べて 48%程度優れた $\mathsf{f}(\mathsf{rd}(s))$ 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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ を下げること」を解決できたことが分かる．

- $\mathsf{f}(\mathsf{rd}(s_{read}))$ の逆転現象
  図 10 を見ると，グラフの変化が単純ではないことが分かる．図 10 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かかわらず，$\mathsf{f}(\mathsf{rd}(s_{read}))$ 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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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，$\mathsf{f}(\mathsf{rd}(s_{read}))$ 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，$\mathsf{f}(\mathsf{rd}(s_{read}))$ 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 項で示される $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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かかわらず，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 11 においても図 10 と同様，グラフの変化が単純でない結果が見られる理由は，図 10 同様に，負荷が限界点に達していないことだと考察する．

5.4.2 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 6** に示す．

表 6 鮮度に関する最良値，最悪値，標準偏差値の範囲
Table 6 Ranges of best, worst and standard deviation value with freshness.

	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 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 倍程度に収まっていることが分かる．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 倍程度あることが分かる．

6. 議　　論

6.1 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，修復作業を行う．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 1 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 台の端末すべ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[14]．有線環境でのビットエラー率は 10^{-9} を下回る[15]．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 端末すべてに届く確率は，次のように計算される[14]．

$$
\begin{aligned}
&1\text{ パケットが } x \text{ 端末に届く確率}\\
&= (1-e)^x\\
&= (1-736\times 10^{-9})^x
\end{aligned}
$$

これより UDP パケットが落ちる確率は高くないから，修復作業は頻繁にはおこらないことが分かる．それゆえ，修復作業により TCP ログサーバがブロックされることで，$\mathsf{f}(\mathsf{rd}(s_{read}))$ が劣化する事態が，本論文での実験において頻繁には発生しなかったと考えられる．

6.2 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 $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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2通りがある．1つはログホストのスレッド処理能力であり，もう1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，$\mathsf{f}(\mathsf{rd}(s_{read}))$ 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，$\mathsf{f}(\mathsf{rd}(s_{read}))$ は大きくなる．

いずれの場合においても $\mathsf{f}(\mathsf{rd}(s_{read}))$ 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．

7. 結　　論

本論文では，連続的なセンサデータストリームをつねに監視する必要があるアプリケーションに対して，優れた鮮度と永続性を保ったデータを提供する手法を示した．提案手法は，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，処理負荷を減らすことだった．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 方式より優れた鮮度を持ち，さらに等しい永続性を持つデータを提供できた．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．

謝辞　本研究の一部には，平成 14 年度情報処理振興協会（IPA）未踏ソフトウェア創造事業未踏ユース「センサデータベース管理システムの開発」（プロジェクトマネージャ竹内郁雄電気通信大学教授）のご支援をいただきました．査読者の方からは有益なコメントを多数いただきました．ここに記して謝意を表します．

参 考 文 献

1) Kohno, M. and Anzai, Y.: An Adaptive Sensor Network System for Complex Environments, *Proc. 5th International Conference on Intelligent Autonomous Systems*, pp.21–28 (1998).
2) 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，信学技報，Vol.103, No.32, pp.7–12 (2003).
3) Bonnet, P., Gehrke, J. and Seshadri, P.: Towards Sensor Database Systems, *Mobile Data Management, 2nd International Conference*, Lecture Notes in Computer Science 1987, pp.3–14 (2001).
4) Carney, D., Cetinternel, U., Cherniack, M., Convey, C., Lee, S., Seidman, G., Stonebraker, M., Tatbul, N. and Zdonik, S.: Monitoring Streams — A New Class of Data Management Applications, *Proc. 28th International Conference on Very Large DataBases* (2002).
5) Madden, S.R. and Franklin, M.J.: Fjording the Stream: An Architecture for Queries over Streaming Sensor Data, *Proc. 18th International Conference on Data Engineering* (2002).
6) Babcock, B., Babu, S., Datar, M., Motwami, R. and Widom, J.: Models and Issues in Data Stream Systems, *Proc. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (2002).
7) Silberschatz, A., Korth, H.F. and Sudarshan, S.: *DATABASE SYSTEM CONCEPTS*, 4th edition, McGraw-Hill (2002).
8) PostgreSQL. http://www.postgresql.org
9) 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，第13回データ工学ワークショップ（DEWS2002）(2002).
10) Hvasshovd, S.-O., Torbjørnsen, Ø., Bratsberg, S.E. and Holager, P.: The ClustRa Telecom Database: High Availability, High Throughput, and Real-Time Response, *Proc. 21th International Conference on Very Large DataBases*, pp.469–477 (1995).
11) Kawashima, H., Toyama, M., Imai, M. and Anzai, Y.: Providing Persistence for Sensor Streams with Light Neighbor WAL, *Proc.*

Pacific Rim International Symposium on Dependable Computing (*PRDC2002*), pp.257–264 (2002).
12) Kawashima, H., Toyama, M., Imai, M. and Anzai, Y.: Providing Persistence for Sensor Data Streams with Temporal Consistency Conscious WAL, *Proc. IASTED International Conference on Information Systems and Databases* (*ISDB 2002*), pp.13–18 (2002).
13) 大崎竜太，上原邦昭：Dynamic Time Warping 法を用いた身体運動の動作識別，情報処理学会データベースシステム研究会研究報告，Vol.116, No.57, pp.13–18 (1998).
14) 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).
15) 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).

(平成 15 年 3 月 25 日受付)
(平成 15 年 7 月 14 日採録)

（担当編集委員　飯沢 篤志）

川島　英之（学生会員）

平成 11 年慶應義塾大学理工学部電気工学科卒業．平成 13 年同大学大学院計算機科学専攻前期博士課程修了．現在同大学院開放環境科学専攻後期博士課程在学中．データベースシステムの研究に従事．ACM，IEEE 各学生会員．

遠山　元道（正会員）

昭和 54 年慶應義塾大学工学部管理工学科卒業．昭和 56 年同大学大学院修士課程修了．昭和 59 年慶應義塾大学工学部管理工学科助手．平成 4 年専任講師．平成 8 年情報工学科に移籍．現在に至る．博士（工学）．平成 8 年オレゴン大学院大学客員研究員．平成 10 年〜13 年科学技術振興事業団さきがけ研究 21「情報と知」領域研究員．主にデータベースの研究に従事．電子情報通信学会，日本ソフトウェア科学会，IEEE Computer Society，ACM 各会員．

今井　倫太（正会員）

平成 4 年慶應義塾大学理工学部電気工学科卒業．平成 6 年同大学大学院計算機科学専攻修士課程修了．同年，NTT ヒューマンインタフェース研究所入社．平成 9 年 ATR 知能映像通信研究所へ出向．平成 14 年慶應義塾大学大学院理工学研究科開放環境科学専攻後期博士課程修了．現在，同大学理工学部情報工学科専任講師，および ATR 知能ロボティクス研究所客員研究員，科学技術振興事業団さきがけ研究 21 研究員．博士（工学）．VR 上のエージェントや自律ロボットとのインタラクションの研究に従事．ロボットとの対話，センサを用いた状況知覚に興味を持つ．電子情報通信学会，人工知能学会，ヒューマンインタフェース学会，日本認知科学会，ACM 各会員．

安西祐一郎（正会員）

昭和 49 年慶應義塾大学大学院博士課程修了．昭和 63 年より慶應義塾大学理工学部教授，平成 5 年より理工学部長．平成 13 年より慶應義塾長．この間昭和 56 年〜57 年カーネギーメロン大学客員助教授．計算機科学，認知情報処理過程の研究に従事．工学博士．電子情報通信学会，日本認知科学会，ACM，IEEE 等会員．