DEIM Forum 2011 B9-6

センサーデータマップにおけるSSD利用COLR-Treeシステムの検討

徳村　力太† 　大森　匡†

† 電気通信大学情報システム学研究科　〒 182–8585 東京都調布市調布ケ丘 1-5-1
E-mail: †{tokumura,omori}@hol.is.uec.ac.jp

あらまし　2 次元平面上に配置されたセンサー集合から送られる観測データ流を問い合わせ可能な形にオンラインで集約変形する時空間インデックスシステムとして, Ahmad らが提案した COLR-Tree システム [1] がある. このシステムは, 指定領域内にいる直近数十分間程度の有効センサーからの観測データ分布を問い合わせる能力を持つ. しかし, 問い合わせで使える時間窓幅が狭い, 長期間の履歴データを保存して過去の一期間のデータを問い合わせる能力がない, もともとハードディスク上の SQL サーバを使っていて高速なイベント処理までは考えていない, などの点が挙げられる. そこで, 本研究では, 永続性を維持したままこれらの諸点を解決するため，データベースリレーションを修正した COLR-Tree システムを SSD 上の関係データベースに試作し, その能力を検討する.

キーワード　センサーデータ処理，Collection R-Tree

A Report on a Modified COLR-Tree System for On-line and Historical Sensor Data Map

Rikita TOKUMURA† and Tadashi OHMORI†

† The University of Electro-Communications, Graduate School of Information Systems
E-mail: †{tokumura,omori}@hol.is.uec.ac.jp

1. はじめに

1.1 研究背景

2 次元上に配置されたセンサー群からの大量データストリームを問い合わせ可能な形にオンラインで集約変形する時空間インデックスシステムとして, Ahmad らによって提案された COLR-Tree(Collection R-Tree) システム [1] がある.

本稿で言うセンサーデータストリーム処理とは, 従来のストリーム処理 [2], [4] ではなくスマートグリッド [3] のように, 広域に配置されたセンサーが 10 分間隔程度でデータを送ってくるようなデータストリーム処理である. この処理は, 上述した COLR-Tree と呼ばれる広域センサー Web ポータルシステムとして成功しており, レストランの待ち時間や水道の流量の時間変化を都市の広領域で追跡するシステムである. ただし, センサーからのデータ生成速度は速くなく, 10 分から 30 分間隔と思われる. ハードディスク (HDD) 上に SQLserver を構築して処理を行っている. 連続問い合わせよりは one-time ad-hoc な集計問い合わせが基本である. 例えば,「指定領域内で 30 サンプル以上使って, 直近 10 分以内で有効なセンサーから発信されているレストランの待ち時間を知りたい」,「指定領域内の 1 マイル四方にある全てのセンサーを 1 点のデータとして集約し, 平均値の平面分布を求めたい」がある.

このように, COLR-Tree システムは直近 10 分間程度の指定領域内の有効センサーからの観測データ分布を問い合わせる能力がある.

しかし, 原論文 [1] で提案されている COLR-Tree は，問い合わせで使える時間窓幅が狭い, 長期間の履歴データを保存して過去の一期間のデータを問い合わせる能力がない, もともと HDD 上の SQL データベースシステムを使って作られているために高速なイベント処理速度を考えていない, などの点が挙げられる．具体的には, COLR-Tree が HDD 上に構築されたことを考えると, 次のような性能上の限界が考えられる：

- HDD 利用のために，イベントを処理して COLR-Tree で扱える速度に限界があり，また，質問できる直近 W 時間の大きさ W があまり長く取れないこと.
- 長期間の履歴データを保存して問い合わせ処理可能にするには向いていない. また, 原論文の COLR-Tree ノードを表すリレーションにおいて指定オブジェクトの過去の履歴追跡問い合わせ (tracking query) を行うと, 2 次インデックスを経由したランダムアクセスが頻発して, 処理が非常に遅くなるだろうということ.

一方で, スマートグリッドの電力メーター管理などを考えると，時空間上の観測データ系列を長期履歴として保存して空間集約問い合わせをアドホックに行う能力は重要である．そこで, 本研究では, これらの諸点を解決することを意図してデータベースリレーションを修正した COLR-Tree システムを，HDD と SSD 上で試作し，センサーデータストリームの集約問い合わせ処理可能なデータベースシステムとしての可能性を検討する. そして、指定領域内の直近 W 時間での集約問い合わせと過去の一期間における指定領域内のセンサー群の履歴問い合わせについて実現できる性能を試す.

以下, 2 章で COLR-Tree システムの概要を述べた後, 3 章では SSD を想定した COLR-Tree システムの設計を示し, 4 章では実験概要を説明して, 5 章でまとめを述べる.

2. COLR-Tree

2.1 COLR-Tree とは

COLR-Tree (Collection R-Tree) は R-Tree を基に構成され, センサーからのデータを効率良くキャッシュし, サンプリングを工夫することで, ユーザーからの時空間上でのセンサーデータ集約問い合わせに答えている. 数階層のノードで構成され, 最下層である葉ノードはセンサーデータを直接格納する. 上位層は下位層の集約したデータ (min, max, avg) を格納している. センサーの故障や, パケットロスによるセンサーデータを報告しないセンサーの事も考慮したり, 2 次元配置されたセンサーを領域ごとに管理しているので, ユーザーからの問い合わせ Q が複数の領域に重なる時のサンプリングを工夫して行っている (図 1 参照). 実装には SQL Server2005 を使い, ホストマシンは 2GB のメモリ, 記憶装置には HDD を使っている. COLR-Tree で扱っているセンサーデータはイベントの発生頻度が高くない. また, データには有効時間を設けてノードに格納しているので, 有効時間が過ぎたデータはノードから削除されるモデルになっている. ノードの詳細については 3.2 節で説明する.

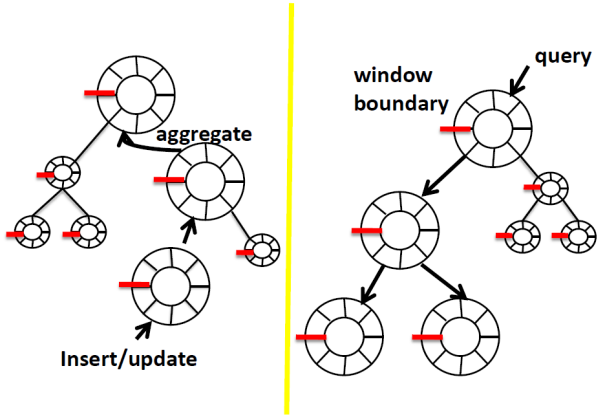

図 1　COLR-Tree
(文献 [1] より抜粋)

2.2 COLR-Tree の適用例

COLR-Tree の実験データにはレストランの待ち時間を発信するセンサーからのデータを代表例として用いている. 対象地域を R-Tree で表現して，370,000 個のセンサーからのデータ (この場合はレストランの待ち時間) を対象として，指定領域内の直近 W 時間のデータ集約値を 2 次元平面上で指定粒度でクラスタリングして表示する.

原論文における COLR-Tree の問い合わせ例を以下に示す. 問い合わせは，ユーザが調べたい領域, 時間, サンプルサイズ等を指定し, 集約を行う. 下の例では，指定領域内に直近 10 分間に有効なデータを送っているセンサーがいくついるかを集計している. 指定領域内のセンサー 30 サンプルをできるだけ一様に選び, 各センサー間の距離が 10 マイル以内にあるセンサーを 1 つのグループとして，各グループごとの有効センサーデータ数を求めている.

```
SELECT count(*)
FROM sensor S
WHERE S.location WITHIN Polygon(< 緯度, 経度 >)
AND S.time BETWEEN now()-10 AND now() mins
CLUSTER 10 miles
SAMPLESIZE 30
```

3. 修正版 COLR-Tree の構成

3.1 本システムの狙い

本システムの狙いは, HDD 上で構築していた COLR-Tree に、指定領域内のセンサー群の長期履歴問い合わせ能力を与えて、インデックス経由のランダムアクセスによる履歴系列問い合わせの処理効率を SSD 利用によって補い、かつ、元の構成よりも高いイベント処理速度を目指すことである.

3.2 Ring Relation

本システムで使う Ring Relation について説明する. COLR-Tree ではノードと呼ばれていたが, 本システムでは Ring Relation と呼び, 説明を行う. Ring Relation も COLR-Tree のノードと同様に時間帯別にセンサーデータを格納するので, スロットという数字で時間帯を管理する. 図 2 に Ring Relation を示す. Ring Relation 内の数字はスロット番号を表している. 数字が大きいスロットほど, 新しいセンサーデータが挿入される. 各領域内にあるセンサーから観測データが間隔をおいて送信される. 時間帯ごとにデータを格納するスロットが変化する. スロットの数が問い合わせの時間窓幅となる. COLR-Tree ではこうした構造のノードで木構造を作っている. 木の階層により扱う領域の大きさが異なり, ユーザーが指定した領域の粒度で問い合わせに答える.

本システムと COLR-Tree とは，古くなったデータを削除せず保存すること, 履歴データ探索用に修正した点が異なる. 図 2 では，履歴データ探索用に, センサー単位で当該センサーの直近の tracking ポインタ (過去のセンサーデータの ID) を持つことで, 最新スロットから過去に遡り, データを探索する. 図 2 に示す赤い点線の矢印 (例えばセンサー ID:i) と青い点線の矢印 (例えばセンサー ID:j) は最新スロットから過去のセンサーデータの履歴探索を行っている様子を表している.

3.3 Historical Relation

古くなったデータを削除せず保存する Relation である Histor-

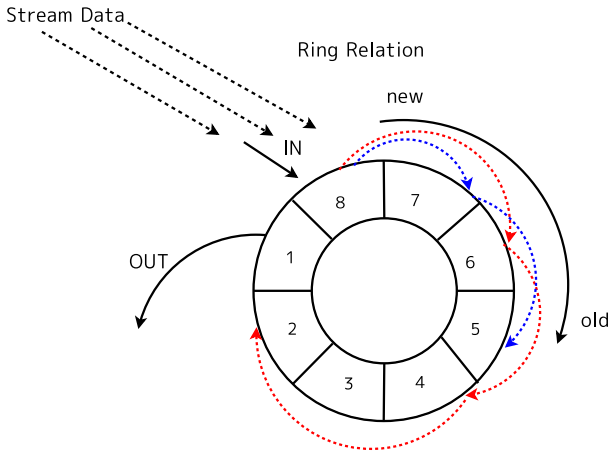

図 2　Ring Relation

ical Relation について説明する. COLR-Tree は Ring Relation 内にあるデータだけを問い合わせることを考えていたが，本システムでは過去のデータを Historical Relation に保存することで問い合わせできるデータを増やした. Historical Relation は Ring Relation と同じスキーマで構成する. スキーマについては 3.6 節で詳細を説明する.

Ring Relation から Historical Relation にデータを移動させる操作はデータベースには重たい処理となる. そのため，今回は，Historical Relation を別に作ってデータを移動するのではなく，新たな Ring Relation を作成し, 古い Ring Relation 自体を履歴データを保存するリレーションとして扱うことにした.

従って，Ring Relation 自体は基本的には追加型のリレーションであり，ログファイルと同じ構造を持たせて SQL 問い合わせ可能にしたことになる.

3.4　修正版 COLR-Tree の全体構成

センサーデータを直接受け取る Ring Relation を葉ノード L_i, また各 L_i からのセンサーデータの集約値を受け取る Ring Relation を Region ノード R_i として話をすすめる. 図 3 に本研究のモデルを示す.

センサーからのストリームデータは, 最初は葉ノード L_i に格納される. 有効時間が過ぎたデータは Ring Relation から削除され, 下位層の Historical Relation に保存される. センサーを配置した対象領域を R-Tree の構造で表し，各葉ノードに対応した領域内のセンサーから送られてきたデータについて，対応する個別の Ring Relation にデータを格納する. 例えば, 図 3 では L_1 領域内のセンサーデータは L_1 に格納する. 上位層のノード R_1 は，当該スロットごとに，L_1から L_3 の各 L_i について，対応するセンサーデータの集約値 (min, max, avg) を格納する. このように，領域と時区間を指定した集約問い合わせが発行されると，なるべく下位層まで行かないように, 上位層のノードに下位の集約したデータをもたせて，問い合わせ処理を行う. (この点は原論文と同じである). 図内の赤い矢印は上位層への集約 (aggregate) と更新 (update) を表す. 黒い点線の矢印は各領域がそれぞれの葉ノードへデータを挿入 (insert) することを表す. 青い矢印は葉ノードから有効時間が過ぎたデータを下位層の Historical Relation へ送ることを表す.

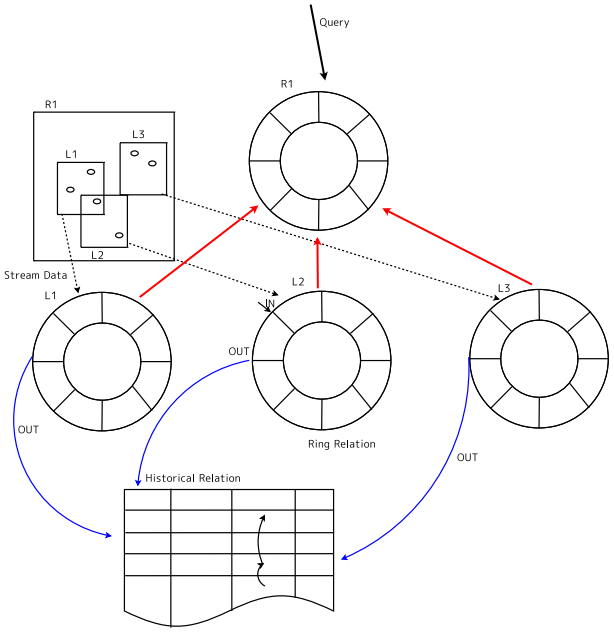

図 3　Ring-Relaltion と Historical-Relation

3.5　スキーマ

実際に作成したテーブルを表 1, 表 2, 表 3, 表 4 を示す. 表 1 は Ring Relation を表すためのスキーマであり, 表 1 の tid では行番号, ts はセンサーから報告されたデータの時刻印, slotid は Ring Relation 内でのスロット番号, value はセンサーのデータを格納する float 型の配列で value[1] に min(最小値), value[2] に max(最大値), value[3] に avg(平均), value[4] に sum(合計値), value[5] に count(カウント数) を格納し、trackid には直前に存在する同じセンサー番号の tid を格納する. インデックスとして, tid を主キー指定して 1 次インデックス, slotid に 2 次インデックスを作成する.

表 2 では新しいセンサーデータが到着する度に, 最新のセンサー ID の表 1 の tid(行番号) と最新スロットを記録するテーブルである. 過去の履歴データを探索するときは表 2 の prev_tid を見て, 得られた prev_tid(行番号) で履歴データの探索を行う. prev_tid に該当する表 1 の tid を探し, 必要なセンサーデータ (value[1]〜[5]) を受け取り, trackid に記載されている値を使い, 同じセンサー ID の過去のデータにアクセスする. センサー数を増やす時には表 2 をハッシュテーブルにして, アクセスを高速化することも考える.

表 3 は各センサーがどの位置にいるかを記録する. 表 4 は各領域の範囲を記録する.

3.6　問い合わせ

本研究では COLR-Tree のように地図上に 2 次元配置で, 長方形に区切った領域内にセンサーを配置したセンサーマップを用いて問い合わせを行う. 考えられる問い合わせとしてはセンサー粒度の問い合わせと, 領域粒度の問い合わせがある. 1 スロット=30 秒で, 100 センサー (センサー ID 1〜100)/L_1, 100 センサー (センサー ID 101〜200)/L_2 とし 1 スロット内に 3 回/セ

表 1　リングテーブル

属性	データ型	索引型	説明
tid	serial	primary key	行番号 (primary index)
ts	timestamp		時刻印
slotid	integer		スロット番号 (secondary index)
sensorid	integer		センサー番号
value	float[ ]		センサーのデータ
trackid	integer		履歴データ追跡用ポインタ

表 2　辞書テーブル

属性	データ型	索引型	説明
sensorid	integer	primary key	センサー番号
prev_tid	integer		行番号
slotid	integer		スロット番号

表 3　センサーの位置テーブル

属性	データ型	索引型	説明
sensorid	integer	primary key	センサー番号
coordinate	point		センサーの座標
regionid	text		領域番号

表 4　領域テーブル

属性	データ型	索引型	説明
regionid	text	primary key	領域番号
area	box		領域 (長方形)

ンサーがデータを送信してくる場合に 1 時間運転してつくった Ring Relation に行う問い合わせについて説明する.

3.6.1　センサー粒度の問い合わせ

COLR-Tree で行える問い合わせとして, センサー粒度の問い合わせがある. ユーザーが指定した領域 Q 内にあるセンサーから S サンプルを一様に選んで, 過去 W スロットまでの時系列データをセンサー単位で表示する.

問い合わせ領域 Q が一つの葉ノード L とのみ重なる場合 (一つの葉ノード L に含まれる場合も含む) はユーザーが指定した S 個のサンプルを $L_i \cap Q$ から取ればよい. しかし, 問い合わせ領域 Q が複数の葉ノードに重なる場合はどの葉ノードからどのくらいサンプリングを行うかを計算しなければならない.

問い合わせ領域 Q が n 個 の葉ノードに重なる場合には, 最初に Q がどの葉ノードと重なっているか調べる. 重なっている葉ノード L_i の面積で, $L_i \cap Q$ の面積を割った値を A_i とする. つまり $A_i = \frac{(L_i \cap Q) \text{ の面積}}{L_i \text{の面積}}$ となる. L_i の保有するセンサー数を W_i とすると, Q 内のセンサーの総数は $TotalQ = \sum_{i=1}^{n} W_i \cdot A_i$ となる. $TotalQ$ から一様に S 個選ぶので, L_i からは $\frac{W_i \times A_i}{TotalQ} \times S$ 個選ぶ.

例として, $L_1 \cup L_2$ のセンサー (200 個) が一様に二次元配置されている状態で, 図 4 に L_1 と L_2 の複数の葉ノードに重なる問い合わせ Q について説明する. Q の四隅の座標を $(8,1),(8,3),(11,1),(11,3)$ とし, サンプル数を $S = 10$, 指定スロットサイズ (窓幅) $W = 5$ とする.

Q は L_1 と L_2 の複数の葉ノードに重なっているので, L_1 と L_2 についてそれぞれ A_i, W_i を調べる必要がある. サンプル数 $S = 10$ に対して L_1, L_2 から 7 : 3 の割合でサンプリングを行う. 図 5 に Q 内に含まれる 12 個のセンサーを示す. 12 個のセンサーのうち, $Q \cup L_1$ の中センサーから 7 個, $Q \cup L_2$ のセンサーから 3 個選んでいる. 選ばれたセンサーについて最新のスロットから $W = 5$ より過去 5 スロットまで時系列データを返す (図 6 参照).

センサーデータの欠落もあり全てのセンサーが毎スロット, センサーデータを報告してくるわけではないので, センサーデータが存在しないスロットも存在する.

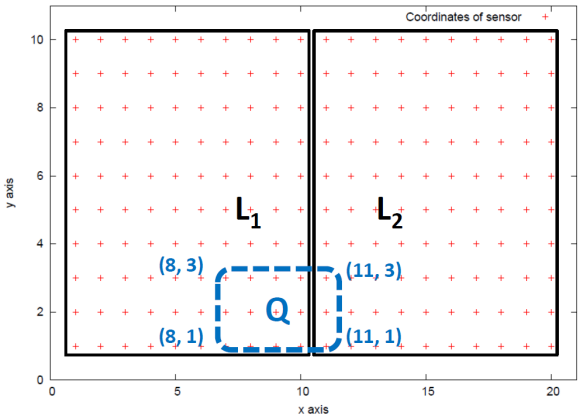

図 4　$L_1 \cup L_2$ のセンサーと問い合わせ領域 Q

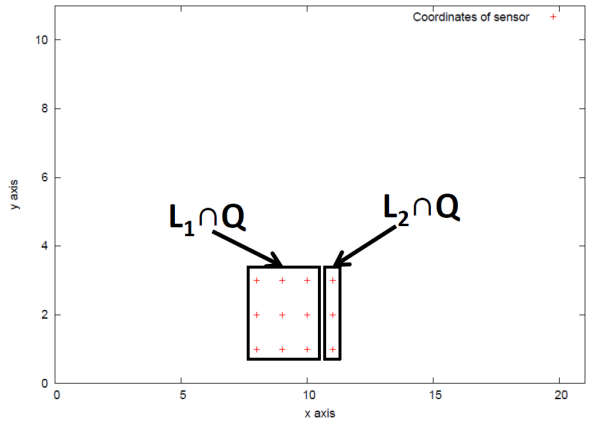

図 5　$L_1 \cap Q$ と $L_2 \cap Q$

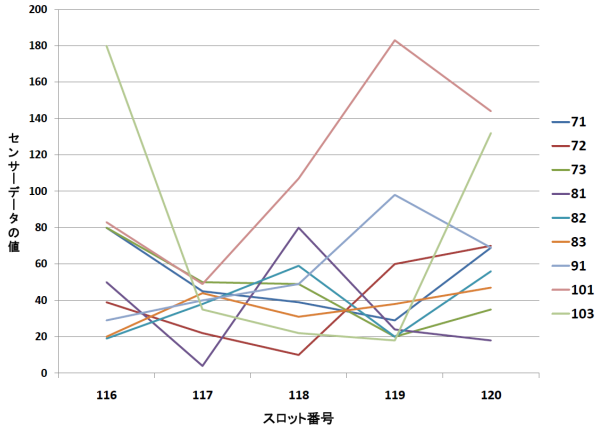

図 6　センサー粒度での問い合わせ

3.6.2 葉ノード粒度の問い合わせ

次に COLR-Tree で行える問い合わせとして, 葉ノード粒度の問い合わせが考えられる. ユーザーが指定した領域 Q 内にあるセンサーから, 過去 W スロットまでの時系列データを葉ノード単位で表示する. 図 7 に問い合わせ結果を示す.

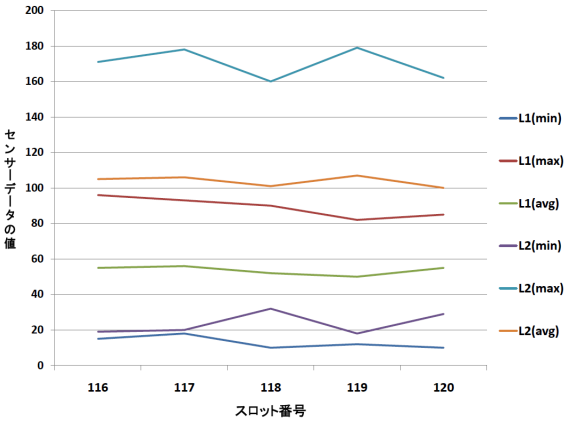

図 7　葉ノード粒度での問い合わせ

4. 実　　験

4.1 実験で用いた COLR-Tree の構成

本研究では, イベント生成プログラムによって擬似的に作成したデータをセンサーデータとして扱い実験を行う. 実験環境のホストマシンの性能は CPU intel core2 Quad 2.40GHz, メモリ 3.3GB, OS は 32bit vine linux5.1, DBMS は PostgreSQL 8.4.4 を使用した. HDD には WDC WD5000AAKS-75VOAO を, SSD には Intel X25-M 80GB SSDSA2MH080G2C1 を使用し, HDD, SSD 共にホストマシンとは SATA インターフェイスで接続を行った. PostgreSQL の共有バッファのサイズは 10MB に設定した.

図 8 に実験で用いた COLR-Tree の構成を示す. 葉ノードは L_1 と L_2 の 2 つで, リージョンノード R_1 はそれらの親ノードである. センサーの総数は 200 個である. L_1 に対応する領域 (図 4 の左半分) にセンサー 1 から 100 の 100 個, L_2 に対応する領域 (図 4 の右半分) にセンサー 101 から 200 の 100 個を配置する. L_1 中のセンサーは座標 (i, j) は $1 \leqq i \leqq 10, 1 \leqq j \leqq 10$ で, L_2 中のセンサーは座標 (i, j) は $11 \leqq i \leqq 20, 1 \leqq j \leqq 10$ である. i と j は整数とする. イベント生成プログラムであるイベントジェネレーター 1 より, クライアントプログラム 1 にはセンサー 1 から 100 のセンサーデータが送られる. イベントジェネレーター 2 よりクライアントプログラム 2 にはセンサー 101 から 200 のセンサーデータが送られる. イベントジェネレーターから, クライアントプログラムへデータを送る時の詳細について説明する. イベントジェネレーターからクライアントプログラムへ 1 回 (1 スロット) で送信するセンサーデータの数は調整ができ, ハッシュ関数 (センサー ID $\bmod n$) で 100 個のセンサーを n 個のグループに分けられ, 各グループごとに送信される. 例えば n の値を 6 にすると, 剰余が $0, \cdots, 5$ まで 6 グループができ, 最初 (1 スロット) は剰余 0 のグループがイベントジェネレーターからクライアントプログラムへ送信され, Ring Relation に挿入をされる. 次 (2 スロット) は剰余 1 のグループがイベントジェネレーターからクライアントプログラムへ送信され, Ring Relation に挿入をされる. 剰余が 2 から 5 までのグループも同様に処理される. n の値が 6 の場合だと, 6 回 (6 スロット) で 100 個のセンサーデータが揃うことになる. 1 スロットの時間はイベントジェネレーターで調整ができ, タイマーで決めた秒数とする.

イベントジェネレーターからクライアントプログラムへの通信はソケットプログラミングにより, TCP でセンサーデータの送受信を行っている. センサーから送信されるデータは [時刻印 (timestamp), センサー番号 (sensorid), センサーのデータ (value)] である.

イベントジェネレーターからセンサーデータを受けとったクライアントプラグラム 1 は葉ノード L_1 にデータを挿入し, クライアントプログラム 2 は葉ノード L_2 にそれぞれセンサーデータを挿入する. この時, 実際に起こりうるイベントとしてセンサーデータの欠落を考慮するが, センサーデータの遅延は考慮しないとものとして実験を行う.

クライアントプログラムから葉ノードにセンサーデータが 1 件挿入される度に L_1 と L_2 は差分計算を行い, 各センサー単位で min, max, avg, sum, count の値を更新していく. つまり最新のスナップショットだけを持つ. 挿入されていないセンサー番号のデータは挿入されるが, 同一スロット内に同じセンサー番号のデータが到着すると, 以前挿入したセンサー番号のタプルを更新していく. スロットが更新されると同様の処理を繰り返す. 上位の Region ノードの R_1 は, L_1 または L_2 のどちらかにセンサーデータが 1 件挿入されると差分計算を行い, 各領域 (L_1 か L_2 か) 単位で min, max, avg の値を更新していく. L_1や L_2 と同様に最新のスナップショットだけを持つ.

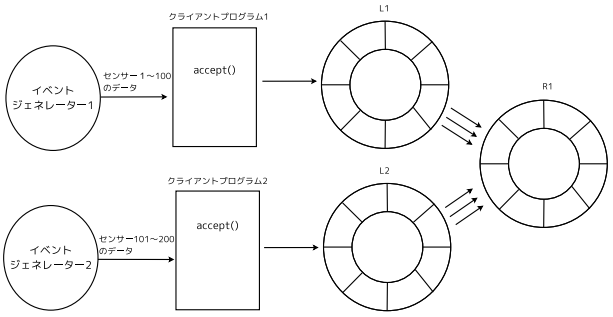

図 8　本システムの構成

4.1.1 ストアドプロシージャ

実験ではセンサー粒度の問い合わせを行う関数をストアドプロシージャとしてデータベースに登録し, 問い合わせを行った. 下記に実験に用いたストアドプロシージャを示す.

- sensor_indexnow('region', toslot, sample)

センサー単位でリレーションの最新スロットから, toslot 個の範囲まで sample 数分のデータをインデックスを使って取りにいく

- sensor_indexmid('region', fromslot, toslot, sample)

センサー単位で fromslot から, toslot 個の範囲まで sample 数分のデータをインデックスを使って取りにいく

sensor_indexnow() は slotid の max を求めてから、指定センサーの trackid を使って必要 slot 範囲を fetch する. これを max で求めた最大のスロット ID から toslot 個の間に指定センサーのデータがある間繰り返す.

sensor_indexmid() は fromslot から toslot 個の範囲にいる指定センサーレコードを見つけてから trackid を使って fetch をする. これを fromslot から toslot 個の間に指定センサーのデータがある間繰り返す.

sensor_indexnow() も sensor_indexmid() も fetch loop の終了は、レコードの slotid を見て判定をする.

4.2 実験概要

実験は過去の履歴データの問い合わせ処理速度を測定するオフライン問い合わせテスト, COLR-Tree へのデータ挿入速度を測定するオンライン運転テストを行った. 表 1 の属性だけでは 1 レコードあたりのデータ量として少ないので, データ量を増やすためにレコードあたり dummy 属性を設け, 256 byte の TEXT データを付け加えて実験を行った.

4.3 オフライン履歴時系列問い合わせテスト

過去の履歴データの問い合わせ処理速度を測定するオフライン問い合わせテストについて説明を行う.

表 5　データセット

table	rows	size
L1 リングテーブル	59999	25MB
L1 辞書テーブル	100	56kB
L2 リングテーブル	59999	25MB
L2 辞書テーブル	100	56kB
R1 リングテーブル	7200	3MB
R1 辞書テーブル	2	48kB

表 5 を 1 セット (1 スロット 3 秒にした時の 3 時間分のデータ) とし, コピーを作成して全部で 4 つのデータセットを用意し, 12 時間分のデータとして扱う. この時, L_1や L_2 の同一スロット内に同一のセンサーが送る回数を 1 回とする. データベース全体としては合計で約 220MB のデータとなった. 問い合わせ領域 $Q = \{Q_1 = '(2,2),(6,2),(2,7),(6,7)', Q_2 = '(5,7),(12,7),(5,10),(12,10)', Q_3 = '(9,1),(16,1),(9,4),(16,4)', Q_4 = '(16,4),(20,4),(16,10),(20,10)'\}$ とし, 4 つのデータセットに対して 4 つの問い合わせ領域 Q を指定して, sensor_indexnow() と sensor_indexmid() を使い問い合わせを行った. 図 9 に問い合わせ領域 Q を示す.

問い合わせの窓幅 W = 1,6, 60, 100, 200, 600 と変化させ, サンプル数を S = 1, 5, 10, 20 にしたときの実行時間を示す. 実行時間は各データセットに 4 つの問い合わせ領域 Q を指定したので, 16 回の平均となっている. どの Ring Relation も直近のスロットが 3600 である. sensor_indexmid の開始点 (fromslot) を 1000 に固定した.

図 10 にサンプル数 S ごとに HDD と SSD の sensor_indexnow()

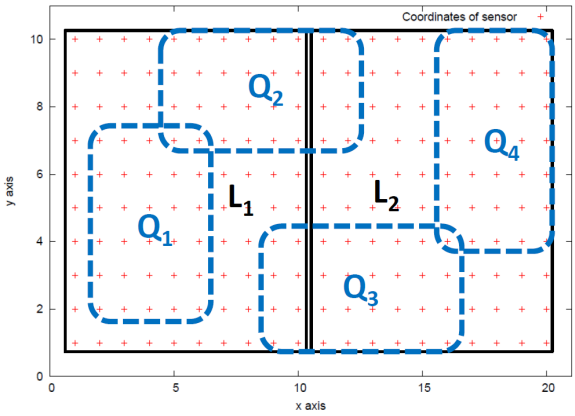

図 9　問い合わせ領域 Q

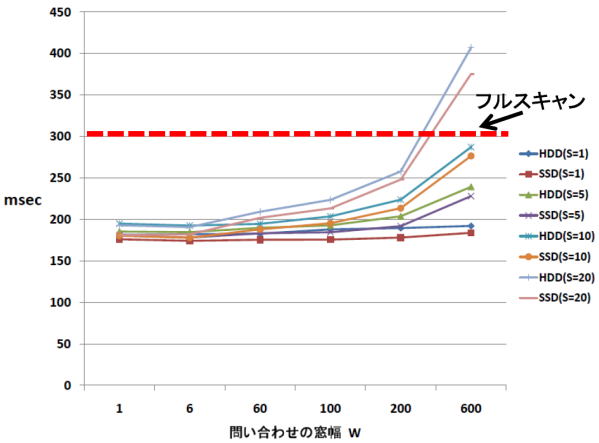

図 10　sensor_indexnow() の実行時間 (msec)

の実行時間を示す. Ring Relation のフルスキャンは約 300msec(赤の破線) なので, オリジナルの COLR-Tree だと 300msec 必要とする. 図 10 より 300msec を越す値は W = 600 で S = 20 のときの HDD と SSD の 2 つである. それ以外は 300msec 以下となり, 本システム独自の trackid を使ったデータアクセスの効果があることが確認できた.

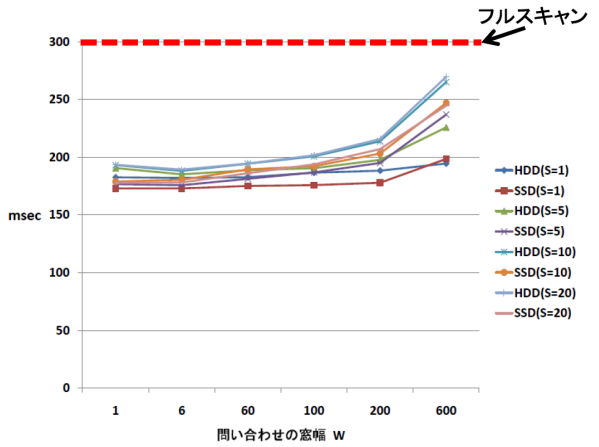

図 11　sensor_indexmid() の実行時間 (msec)

図 11 にサンプル数 S ごとに HDD と SSD の sensor_indexmid() の実行時間を示す. sensor_indexmid() は Ring Relation のフルス

キャンである 300msec(赤の破線) を越す値はなくフルスキャンより速いことが確認できた. sensor_indexmid() でも trackid を使ったデータアクセスの効果を示すことができた.

4.4 オンライン運転テスト

COLR-Tree へのデータ挿入速度を測定するオンライン運転テストについて説明する.

1 スロットの時間を 1 秒に固定し, 1 スロットあたり Ring Relation(L_1や L_2) にデータを送信するセンサー数を変化させてセンサーの挿入に成功したレコード数を HDD と SSD で測定する.

センサー数を 50, 60, 70, 80, 90, 100 と変化させたときのレコード数を確認する. オンラインの運転時間は 10 分とする.

図 12 はセンサー数を増やしたときに挿入に成功した割合を表す.

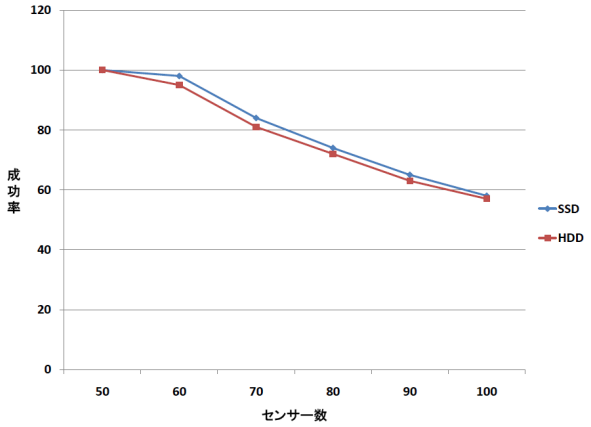

図 12 成 功 率

わずかながら SSD が単位時間あたりに多くのイベントを処理できることを確認した.

4.5 検証と議論

図 13 に共有バッファサイズを変え, 1 スロットの時間を 1 秒, 1 スロットにデータを送信するセンサー数を 50 に固定し, 1 時間運転したときに挿入できたレコード数を示す. 共有バッファサイズが 10MB の時は HDD も SSD も変わらない. しかし共有バッファサイズを 512kB に変更すると差があらわれ, HDD に比べて SSD の方が約 1.2 倍多くレコードを挿入している.

共有バッファサイズを調整してやることによって, データサイズが大きくないときでも HDD と SSD の性能差が出ることを確認した.

このことからもイベント入力速度をあげていくためには, 共有バッファサイズを調整が必要になり, 今回のテストのように共有バッファサイズを 10MB で固定した場合でも Ring Relation を増やすなどスケールアップしていくと差がでると予想できる.

5. おわりに

本稿では，2 次元上に配置されたセンサー群からの観測データ流を集約して時空間問い合わせ可能にする COLR-Tree システムを対象に，長期履歴データへの集約値系列の問い合わせ能

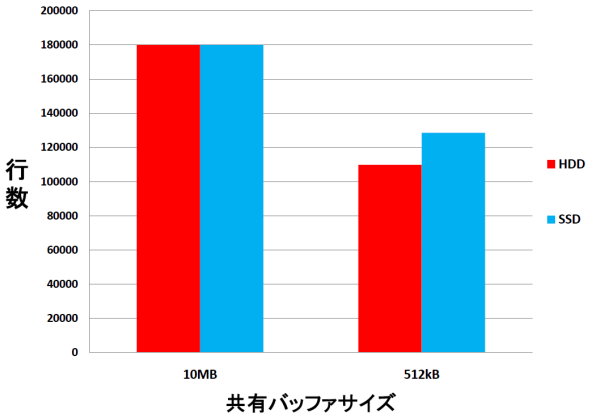

図 13 共有バッファサイズを変更したときの影響

力を付与し，それに伴って中間ノードとなるリレーションの設計を SSD 利用を意図して修正した．その上で，過去の時空間系列の問い合わせ処理，直近 W 時間の時空間問い合わせにおける窓幅 W の長時間化，処理可能なイベント取込速度を調べた．葉ノード 2 つ，中間ノード 1 つからなる最小構成の COLR-Tree を SSD 及び HDD 上の Postgres で試作して基本的な動作効率を調べた．長期間の時系列データ探索は，追加型リレーションで時系列データを表す限り，インデックス経由のランダムアクセスの多発を意味するため，実質的に，該当するノードを表すリレーションのスキャンが HDD 上では選択され易い．一方，SSD のようにランダム読み出しが速いなら，今回用いた trackid を使った系列探索は，センサーデータの欠落にも対処できることもあり，自然な選択である．Ring Relation あたりのセンサー数の増加の扱い方も含め，試作では探索開始位置の決定そのものに時間がかかったため，Ring Relation に与えるべき探索構造には改良の余地は大きい．一方，イベントの処理速度については，今回の構成では最小構成の COLR-Tree と Postgres の共有バッファの性能調整の都合上、HDD/SSD 間でほとんど差が出なかった．ノード数を増やして COLR-Tree へのデータ挿入と直近 (または履歴) 時系列問い合わせを安定した処理速度で扱うことも重要である．

文 献

[1] Yanif Ahmad and Suman Nath “COLR-Tree: Communication-Efficient Spatio-Temporal Indexing for a Sensor Data Web Portal, ” *ICDE*, pp.784-793, 2008.

[2] Arvind Arasu, Brian Badcock, Shivnath Babu, Mayur Datar, Keith Ito, Rajeev Motwani, Itaru Nishizawa, Utkarsh Srivastava, Dilys Thomas, Rohit Varma, and Jennifer Widom. “STREAM: The Stanford Stream Data Manager,” IEEE Data Eng. Bull. vol.26 No.1 pp.19-26, 2003.

[3] 「スマートメーターの賢い使い方」『環境ビジネス』, pp.18-27, 日本ビジネス出版 (2009 年 10 月号).

[4] 北川博之, 川島英之, 天笠俊之, “センシングデータ処理基盤技術-ストリームデータ処理-,” 情報処理学会誌, pp.1119-1126, Vol.51 No.9, 2010.