4L-01

インメモリ・データ圧縮処理を備えた大規模センサネットワーク用分散型データベース

安達 成一† 宮崎 敏明†

†会津大学大学院コンピュータ理工学研究科

1 はじめに

近年、センサネットワーク(SN)は環境情報を取得するために広く使われている。我々は、複数の SN を統合した大規模 SN に於いて、リアルタイム時空間検索を可能にする分散型 DB システム (以後、従来システム)を提案した[1]。当該従来システムは、Geohash[2]と Cassandra[3]を用いて実現した。ここで、Geohash とは、Geocoding の一手法であり、緯度・経度の二次元情報を、一次元の文字列で表し、その長さ（以後、Geohash 長）によって矩形空間の広さを表現するものである。しかし、膨大なセンサデータを管理するに当たり、ストレージ量の増大、処理速度の低下が課題となった。本問題を回避する一手法としてデータ圧縮の導入がある。文献[4]では、Gamma 符号化と固定長を組み合わせることにより、値が高変動するセンサデータでも効果に圧縮できる手法を提案している。しかしながら、当該圧縮処理を、各センサやゲートウェイで実行するとその処理の重さからリアルタイムアクセスが困難となる。

本稿では、従来システムの前処理として、文献[4]のセンサデータ圧縮処理を、従来システムを実装しているマシン上で、インメモリ処理することにより、データのリアルタイムアクセスを犠牲にすることなく、ストレージ量の削減に寄与する分散型 DB システムを提案する。

2 提案分散型 DB システム

提案する分散型 DB システムの全体図を図 1 に示す。各 SN から生成されたセンサデータは、インメモリ圧縮処理するためにバッファメモリ（以後、単にバッファ）上に保存される。バッファ内では、センサデータの集合を 1 分毎に区切られたリングメモリ構造で管理する。1 分前の区画内のセンサデータを圧縮して Cassandra へ保存すると共に、2 分前の区画内のセンサデータは削除する。よって、バッファ内には最大で 2 区画(120 秒)分のセンサデータが蓄積される。また、バッファの容量制限を超えないように、データ圧縮と Cassandra への書き込みは 1 分以内に完了する必要がある。また、センサデータは範囲検索可能なように、2 つのソートされたマップの入れ子構造 Map<Key,Map<long,double>>で管理される。換言すると、センサデータは、センサの種類と Geohash 長 12 からなる Key に紐付けられたセンサ値と取得時間から構成される Map<long,double>で保存される。バッファ内のセンサデータに対して時空間検索を行う場合、まず、センサタイプと任意の Geohash 長による前方一致で、その矩形空間内に存在する時系列データのマップを取得する。次に、取得したマップ内で時間範囲検索を実行することにより、所望センサデータを得る。

観測者が、DB システム内に保持されているセンサデータにアクセスする場合、サーバから圧縮書き込み完了時間を取得し、所望のセンサデータの取得開始時間が、その時間の前なら、まだ Cassandra に書き込まれる前なので、バッファにアクセスし、後なら Cassandra へアクセスして所望のセンサデータを取得する。

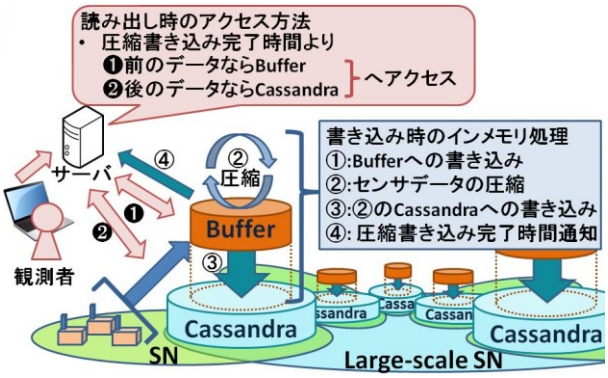

図 1 提案分散型 DB システムの全体構成

2.1 圧縮手法

圧縮する入力データ列はセンサデータ値列と取得時間データ列の 2 つである。実際の圧縮は、センサデータ値列には処理Ⅰ-Ⅴを、取得時間データ列には処理Ⅲを除いた処理Ⅰ-Ⅴを s=0, $p_i = r_i$として適用して行う。

I. 入力データ列$[d_i]_{i=0}^{n}$を整数列$[d'_i]_{i=0}^{n}$に変換する。

$$d'_i = d_i \times 10^s$$

II. 整数列$[d'_i]_{i=0}^{n}$から n − 1個の残差数列r_iを生成する。

$$r_i = d'_i - d'_{i-1} \quad (1 \le i \le n)$$

III. 各残差r_iを正整数 p_iへ写像する。

$$p_i = \begin{cases} 2\,r_i & (r_i > 0) \\ 2|\,r_i| + 1 & (r_i < 0) \\ 1 & (r_i = 0) \end{cases}$$

IV. 各 p_iを n ビット目で分割する。下位ビットは固定長とし、上位ビットのみ 1 ビット加算し Gamma 符号化後のビット数を算出する。$0 \le n \le 32$において総ビット数が最も小さい分割位置を k として決定。

V. 各 p_iを k ビット目で分割し、上位ビットは Gamma 符号化、下位ビットは固定長（入力値そのもの）とし圧縮結果を得る。

解凍に必要な情報はセンサデータ値列の d'_0, s, k, 取得時間データ列の d'_0, k の 5 つである。ここで、Ⅳにおいて、1 ビット加算するのは上位ビットが 0 の場合 Gamma 符号では表現できないためである。

2.2 Cassandra データ構造

図 2 に CQL(Cassandra Query Language)形式によるデータ構造を示す。row_key_index では Geohash 長 5～10 の矩形空間内に存在するセンサの圧縮された時系列データのインデックスを管理し、compressed_data_points では各センサの圧縮された時系列データが格納されている。解凍に必要な情報である取得時間列の d'_0は timestamp へ保存され、センサデータ値列の d'_0, s, k, 取得時間列の k は 64 ビットの thawing_info の中に保存される。

Distributed Database System Using In-Memory Data Compression for a Large-Scale Sensor Network
†Seichi Adachi, †Toshiaki Miyazaki
†Graduate School of Computer Science and Engineering, The University of Aizu

```
CREATE TABLE IF NOT EXISTS dasn.row_key_index (
key text,
head_time timestamp,
compressed_data_points_key text,
PRIMARY KEY (key, head_time, datapoints_key )
) WITH CLUSTERING ORDER BY (
 head_time DESC, datapoints_key DESC );
CREATE TABLE IF NOT EXISTS dasn.compressed_data_points (
key text,
compressed_head_time timestamp,
compressed_time blob,
compressed_sensor_value blob,
thawing_info bigint,
PRIMARY KEY ( key, timestamp )
) WITH CLUSTERING ORDER BY ( timestamp DESC );
```

図 2 Cassandra 内データ構造(CQL 形式)

実験

従来システムと提案システムのアクセス時間とストレージ量の関係を比較した。実験に使用したマシンは、Dell Vostro460(CPU:Intel Corei7-2600 3.4GHz, Memory:8GB, OS:Centos6.5)である。

● 評価用センサデータ

評価に用いたセンサデータ値列$[d_{i,v}]_{i=0}^{n}$は、文献[4]で用いている式$d_{i,v} = m + a_1 \sin 2\pi f_1 i + a_2 \sin 2\pi f_2 i + a_3 X$を用いて生成した。$m$, a_1, a_2, a_3, f_1, f_2は定数パラメータである。また、X はノイズ成分であり、正規分布 $N(0,v)$に従う確率変数とした。ここでは、$m = 0, a_1 = 1000, a_2 = 100, f_1 = 0.0005, f_2 = 0.05, v = 1$とし、ノイズ振幅を$a_3 = 1000$とした。さらに、浮動小数点列を整数列として、DB で保持するために一定倍率を乗ずるが、その変換倍率は、予備実験の結果を踏まえ10000とした。取得時間データ列$[d_{i,t}]_{i=0}^{n}$は$d_{i,t} = \mathrm{UnixTime} + 1000\mathrm{i} + \mathrm{rand}(0,999)$により生成する。rand(0,999)は0~999のランダムな値を生成することを意味する。

● 書き込み時間とストレージ量の比較

12 万センサデータを 1 秒間隔で 1 万回書き込み、その平均書き込み時間とストレージ量の計測を行った。書き込み時間の実験結果を図 3 に示す。提案システムの書き込み時間は 888ms であった。これは、従来システム(1558ms)の 1.8 倍の速さであり、毎秒生成される 12 万センサデータを 1 秒以内、すなわち、リアルタイムに DB に書き込める性能があることを示している。本性能は、インメモリ・データ圧縮のために設けたバッファへの書き込み速度が高速なことが寄与している。バッファに格納された 7.2M データ(=12×10^4×60s)の圧縮処理+Cassandra への書き込み時間の合計は、9340ms であった。Cassandra への書き込みは、前述したように 1 分毎に行うことから、内部処理時間も十分リアルタイム処理できる性能であることがわかる。表 1 は、本実験で使用したストレージ量の総計である。従来システムに比べ、ストレージ量を 78%(=((11.1GB-2.4GB)/11.1GB)×100)削減できている。

● 読み出し時間の比較

毎秒 3 万センサデータを 24 時間分(総計 2.592G データ)書き込んだ後、100 センサから t 秒間のセンサデータを取得する際の読み出し時間の計測を行った。実験結果を図 4 に示す。具体的に、取得時間間隔 t は、t = {1, 10, 100, 1000, 10000}の場合を評価した。提案システムの DB には 60 秒間の圧縮された時系列データが 1 カラムに保存されている。よって、例えば 10 秒間のセンサデータを取得する際、最大 2 カラム分の圧縮されたデータを読み出す必要がある。本事実を踏まえ、上記、各読取時間間隔 t に対して、DB から読み出すカラム数 u を、u={1, 2, 3, 18, 168}とし、計測した読み出し時間は、読み出した各カラムの解凍時間も含め、実際に要求したデータを取得するまでの時間である。また、図中、”提案システム(バッファ)”とあるのは、読み出し対象データがバッファに存在する場合、すなわち、過去 2 分以内に書き込んだデータの読み出し時間である。前述したように、バッファには、2 分(120 秒)間のセンサデータしか存在しないため、t={1,10, 100}の場合のみ計測した。図から分かりように、バッファ上からのデータ読み出しは高速で、158ms であった。取得時間間隔が 10 秒以下の場合、提案システム(DB)は従来システムより最大 29%読み出しに時間を要しているが、その遅延は高々1 秒以内であった。一方、100 秒以上の場合、30%以上高速な読み出しが可能であった。

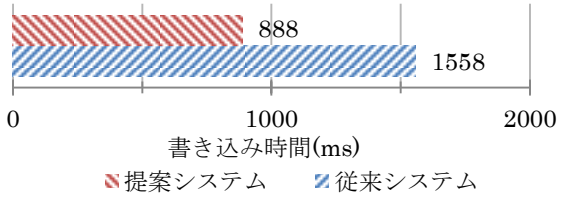

図 3 書き込み時間比較

表 1 ストレージ量比較

	従来システム	提案システム
ストレージ量	11.1GB	2.4GB

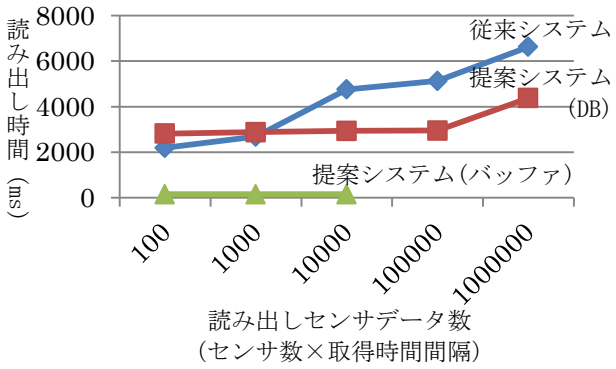

図 4 読み出し時間比較

3 結論

本稿では、前処理としてデータ圧縮をインメモリ処理で実行する分散型 DB システムを提案し、筆者等が過去に提案した分散型 DB システムに比べ、データアクセス時間を大幅に増加させることなく、高変動なセンサデータでも、ストレージ量を 78%削減ができることを示した。また、安価なサーバマシン環境でも、12 万センサデータを 1 秒以内に格納可能なことを示した。

謝辞

本研究の一部は、総務省戦略的情報通信研究開発推進制度(SCOPE No. 121802001)の支援を受けて実施したものである。

参考文献

[1]T. Miyazaki, N. Suematsu, D. Baba, P. Li, S. Guo, J. Kitamichi, T. Hayashi, and T. Tsukahara, "Demand-Addressable Sensor Network: Toward Large-Scale Active Information Acquisition," IEEE Sensors Journal, vol. 16, no. 20, pp. 7421-7432, October 2016, DOI:10.1109/JSEN.2016.2575846
[2]Geohash, http://geohash.org
[3]Apache Cassandra, http://cassandra.apache.org/
[4]柴田秀哉, 高山茂信 “高変動時系列整数データ圧縮手法” D6-3, DEIM Forum (2015)