

大規模センサデータに対する柔軟な管理法の提案

岩田寛生[†] 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

複数のセンサネットワークから生成されるデータを効率的かつ安定して管理する手法が求められている。特に無線センサネットワーク (WSN) は、個々のセンサノードの動作が一般に不安定であり、ネットワークトポロジも頻繁に変化する。よって、複数の WSN から生成される膨大かつ不安定なデータをリレーショナルデータベース (RDB) など従来型のデータベース (DB) で管理することは必ずしも得策ではない。本稿では、分散 DB に基づく大規模センサデータの柔軟な管理法を提案する。

2. WSN におけるデータ管理

センサから得られる膨大なデータの管理、検索に関して様々な提案がなされている。例えば、異種センサネットワーク環境を想定したセンサデータの位置情報による統合手法[1]や異種センサデータを複数の DB ピア (以下、単にピア) で管理、検索するために PostgreSQL を改良した分散 DB TomuDB[2]などがある。TomuDB はセンシング領域や粒度情報に基づいた効率的な検索が可能であるが、ピア同士の接続性がツリー構造のため単一障害点 (Single Point of Failure) が存在するという問題がある。また、既存技術は、RDB を用いたものが主である。しかし、RDB は事前に決めたスキーマに従ってデータを保管する必要があり、新規センサノードの追加など、動的に保管すべきデータが変化する WSN のデータ管理には適さない。一方、NoSQL と呼ばれる分散 DB、例えば Hadoop[3]、mongoDB[4]、Cassandra[5]などが、ビッグデータの管理手法として近年注目されている。Hadoop と mongoDB はマスタ/スレーブモデルを採用しており、単一障害点が存在するため、本稿で目指すピアも含めた柔軟な管理という点において問題が残る。一方、Cassandra はピア・ツー・ピア分散モデルを採用しており、複数のピア同士でリング状のクラスタを構成するため単一障害点がない。また、Cassandra は列指向 DB であり、動的にカラムを追加してデータを管理するため WSN におけるデータの動的な変化には、対応するカラムを追加するのみでよく、柔軟に対応することが出来る。さらに、構成したクラスタ間でレプリケーションを行うことで、データを多重化し、耐障害性も確保できるという特徴がある。

本稿の目標は (1) WSN からのデータを安定に収集・管理出来ること、および (2) WSN の環境変化にも柔軟に対応出来ること、の2点である。

3. データ管理の検討

2章で述べたように、Cassandra は単一障害点を持たず同一データを多重化して保管出来るためピアの耐障害性を持ち、列指向 DB であることからデータの追加、更新、削除にも柔軟に対応出来る。本稿では、複数の WSN で Cassandra クラスタを構成し、Cassandra のスケールアウトし易さと柔軟なデータ追加機能を利用した大規模 WSN のデータ管理法を提案する。図 1 にシステム概要を示す。

An approach to flexible management method for huge sensor data

[†]Hiroki Iwata, [†]Toshiaki Miyazaki

[†] Graduate School of Computer Science and Engineering, The University of Aizu

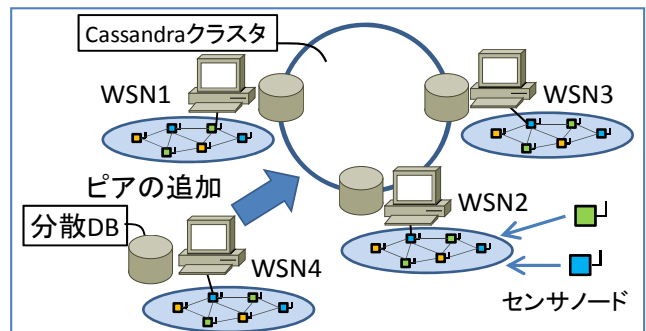


図 1 システム概要

各 WSN を管理し、センシング情報を保管する分散 DB を 1 つのピアとし、Cassandra クラスタを構成する。また、各分散 DB のロウキーやカラムをセンサノードと考えることで、センサノードの追加・削除にも柔軟に対応できる。

3.1 複数の WSN の統合

Cassandra クラスタへ新規ピアを追加する際には、初回にクラスタ名といずれかの IP アドレスを設定するだけでよい。初回以降は Cassandra クラスタ内で Gossiping により他のピアと通信し合いリング状の構成を自動的に保つため、初期設定したピアが故障しても、運用上その影響は受けず、複数の WSN の統合も容易である。ただし、クラスタへの新規ピアの追加によってハッシュ値のバランスが均等ではなくなるので、式(1)に従いピアのハッシュ値の再設定を行う。

$$\text{各ピアのハッシュ値} = 2^{127} / \left(\frac{\text{ピアID}}{\text{ピア総数}} \right) \quad (1)$$

3.2 データ管理

Cassandra はデータを追加、更新、削除をする際にデータから一意に決まる 0 から 2^{127} までのハッシュ値を用いて、どのピアに保管されるかを決める。更新、削除は既存データに削除フラグを付け、新規データの登録は、当該ピアに追加することで実現する。WSN のゲートウェイをピアとした場合、センサノード数と追加データ量は比例関係にあり、その増減に従って、データは各ピアに均等に保管される。また、新規センサノードが増加された場合に対しても、Cassandra のスキーマレスな特徴から、RDB の様にスキーマを定義し直すことなく、ロウキーやカラムを追加するだけで柔軟に対応出来る。また TomuDB の様に複数の WSN を統合した場合、あるピアにデータがまとまって保管されているとアクセスがそのピアに集中してしまうことがある。しかし、Cassandra を用いて各ピアのデータ量を均等にすることで負荷分散をすることが出来る。

3.3 処理速度とデータ一貫性

Cassandra では処理速度とデータ一貫性の維持にトレードオフが存在する。具体的には、レプリケーション数、書き込み、読み込みの際の設定において、書き込み数 (W)、読み込み数 (R)、保存ピア数 (N) としたとき、 $W + R > N$ であれば強一貫性となり、 $W + R \leq N$ であれば結果整合性となり処理速度が高くなる。結果整合性とは一貫性を保つための排他制御を行わずピア間の通信遅延による一時的な一貫性の欠如に関しては許容をする

表 1 実験環境

	ケース 1	ケース 2	ケース 3
試行回数	100	100	100
センサノード数	1000	10000	100000
センサノード情報数	10	10	10

表 2 ロウキーとカラムの構成が異なる 2つのパターン

	ロウキー	カラム
パターン 1	試行回数+ センサノード数	センサノード情報数
パターン 2	試行回数+ センサノード情報数	センサノード数

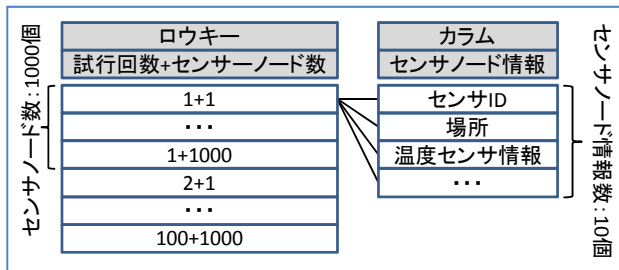


図 2 パターン 1 かつケース 1 の場合のロウキーとカラムの構成例

いうものである。WSN への要求によって処理速度と一貫性のどちらを重視するか、上記適宜パラメータを設定することが出来る。

4. 検証

4.1 実験環境

同一ネットワーク上に設置した 1 台の検証用 windows 機と 3 台の windows 機上の仮想 linux 機に、環境構築時点で最新バージョンである Cassandra 1.1.6 を用いてクラスタを構築した。また、レプリケーション数を 2 とし、ハッシュ値は事前に均等になるように設定して実験を行った。検証用 windows 機はデータ追加のために用いた。

4.2 実験

表 1 に示したセンサノード数の異なる 3 つのケースに対して、表 2 に示したロウキーとカラムの構成が異なる 2 つのパターンを適用し、各パターンにおいてカラムとロウキーの肥大化による影響を read/write の速度、データの均等性の観点から検証した。ここで、センサノード情報数とは、一つのセンサノードの持つ ID などの管理情報とセンシングデータを合わせた、センサノード当たりで管理すべきデータの個数であり、本実験では 10 個と固定した。図 2 にパターン 1 かつケース 1 の場合のロウキーとカラムの構成例を示す。本例から容易に理解できるように、パターン 1 ではセンサノード数が増加するとロウキー数が増加する。一方、パターン 2 の構成では、カラムをセンサノード数に対応させるため、センサノード数が増加するとカラム数が増加する。read 速度の値は、異なるロウキーに対して 100 回アクセスを行い、その read 速度の平均値を用いた。

4.3 実験結果

write 速度の結果を表 3 に示す。パターン 1, 2 とも総レコード数の増加に比例して書き込み時間も増加する。また、総レコード数は等しいにもかかわらずロウキー数の多いパターン 1 とカラム数が多いパターン 2 で write 速度に 10 倍以上の差異があることが分かる。これは、複数のロウキーを作成するオーバーヘッドの影響と思われる。表 4 に read 速度の結果を示す。表から分かるように、全体的に Cassandra に書き込まれるレコード数の増加に

表 3 write 速度 (単位: ms)

	ケース 1	ケース 2	ケース 3
パターン 1	424593	4404391	45435813
パターン 2	34813	327610	3654875

表 4 read 速度 (単位: ms)

	ケース 1	ケース 2	ケース 3
パターン 1	4.092	2.421	56.036
パターン 2	2.994	5.478	44.228

表 5 各ピアに格納されているデータ量 (単位: MB)

パターン	ケース	ピア1	ピア2	ピア3	総データ
1	1	5.30	5.28	5.28	15.86
	2	54.54	54.51	54.52	163.57
	3	560.16	560.17	560.17	1680.5
2	1	3.28	3.06	2.48	8.82
	2	33.66	33.49	32.69	99.84
	3	340.65	339.64	325.55	1005.84

伴い検索に時間が増大し、データの read 速度は遅くなる。また、ケース 3 では両パターン共に read 速度が著しく低下している。これは、データ総量が大きく、他ピアに格納されているデータを read する回数が増加したが、他ピアのデータ read 速度は自ピアのそれと比べて遅いことが要因である。表 5 に各ピアに格納されているデータ量を示す。表から分かるように、データの均等性の観点からは、パターン 1 の方がパターン 2 よりも各ピアのデータのバラつきは小さい。一方、総データ量はいずれのケースにおいてもパターン 2 の方が少なく、ディスクの使用効率の点ではパターン 2 の方が良いという結果になった。

以上、read/write 速度とデータの均等性の実験結果から、カラム数が多い構成にすることで write/read 速度は向上しディスクの使用効率も改善するということが分かった。

5. 結論

本稿では、Cassandra のスケールアウトし易さと柔軟なデータの追加機能を利用して、WSN の柔軟な管理の際に有効なロウキーのパターンに関して比較、検討を行った。実験の結果、多数ロウキーと少数カラムの構成よりも、少数ロウキーと多数カラムであるワイドロウ構成の方が read/write 速度、データの保存効率という面で良好であることが分かった。今後は、多数のピアでクラスタを構成し、大規模 WSN を想定した実験を行う予定である。

謝辞

本研究は、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] 白石陽, 安西祐一郎, センサデータの視覚化のためのインクリメンタルな空間集約手法, 情報処理学会論文誌: データベース, Vol. 45, No. 7, pp. 63-76, Jun 2004.
- [2] 岩井将行, THEPVILLOJANAPONG Niwat, 石塚宏紀, 中村陽一, 金井圭介, 白石陽, 戸辺義人, TomuDB: 都市空間センサノード情報を扱うデータベースシステム, 電子情報通信学会技術研究報告. USN, ユビキタス・センサネットワーク, Vol. 108, No. 138, pp. 13-18, July 2008.
- [3] Hadoop, <http://hadoop.apache.org/>
- [4] mongoDB, <http://www.mongodb.org/>
- [5] Cassandra, <http://cassandra.apache.org/>