CD-002

リアルタイムセンサデータに対するストリーム処理アーキテクチャの検討
A Study of Stream Processing Architecture for Real-time Sensor Data

松山 拓紀† 大西 直哉†
Hiroki Matsuyama Naoya Ohnishi

1. はじめに

近年，社会インフラ向けの制御システムにおいて，制御装置に付随するセンサから生成されるデータを利活用することが注目されている[1]．利活用の一例である故障予測や最適化制御では，広域に存在する制御システムのセンサデータを長時間収集し解析に用いるため，インターネットを介して接続されるクラウドサーバに膨大なセンサデータを保存する必要がある．しかし，膨大なセンサデータをクラウドサーバに保存するには，通信帯域の圧迫やデータ保存コストの増加という課題がある．

そこで筆者らは，制御装置の近傍にエッジサーバを配置し，クラウドサーバとエッジサーバでデータを分散して保存する，データ分散配置システムを提案している[2], [3]．提案のシステムでは，エッジサーバにセンサデータを保存しクラウドサーバに一次的な解析処理に必要な統計データを保存するため，クラウドサーバとの通信データ量とクラウドサーバでの保存データ量が削減できる．センサデータは常時生成され続けるため，統計データの作成やデータの保存といった一連の処理をリアルタイムに実行する必要がある．

本稿では，エッジサーバにおいてリアルタイムに統計処理と保存処理を行うストリーム処理アーキテクチャを提案し，処理に要する時間の評価を行った結果について述べる．

2. データ分散配置システム

図 1 に，データ分散配置システムの概略を示す．提案しているシステムはセンサ，制御装置，エッジサーバ，クラウドサーバから構成される．エッジサーバとクラウドサーバはデータを保存するためのデータベース(DB)を持つ．

エッジサーバはセンサデータの受信と保存，クラウドサーバの解析で必要となるデータを作成する統計処理，統計処理の結果である統計データの送信を行う．エッジサーバの DB にはリアルタイムに生成される全てのセンサデータを保存するため，保存に要する処理時間が短いドキュメント DB を適用する．

クラウドサーバは統計データの受信と保存，統計データによるデータ解析を行う．クラウドサーバの DB にはデータ解析において複雑なクエリでデータを参照する可能性があるため，RDB (Relational Database)を適用する．統計データによる一次的なデータ解析の後，詳細なデータ解析を行う際はエッジサーバのドキュメント DB からセンサデータを取得する．

エッジサーバとクラウドサーバ間の通信プロトコルは，短いメッセージの頻繁な送受信に適した軽量プロトコルである MQTT を用いる．エッジサーバは，クラウドサーバにおけるデータの識別子となるトピックを統計データに付与してクラウドサーバに送信(Publish)する．クラウドサーバは MQTT を配信するブローカと MQTT を受信するクライアント(Subscriber)を持つ．Subscriber は定められたトピックからメッセージを受信(Subscribe)し，SQL 文を生成し統計データを RDB に保存する．クラウドサーバは，トピックに基づいてデータ分類を行うことで，速やかなデータ分類および保存ができる．

† 株式会社東芝
TOSHIBA CORPORATION

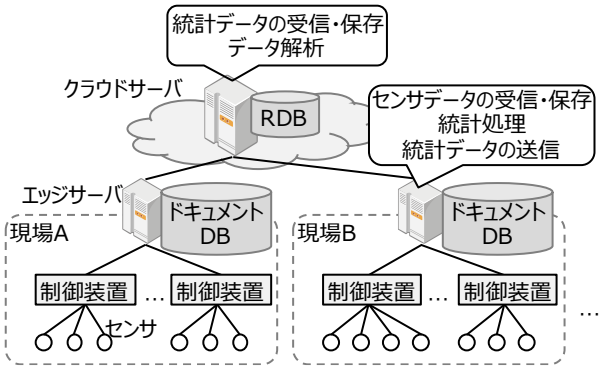

図 1 データ分散配置システム

3. ストリーム処理アーキテクチャ

本章では，データ分散配置システム内のエッジサーバにおける，常時生成されるセンサデータに対して統計処理と保存処理をリアルタイムに行うためのストリーム処理アーキテクチャについて述べる．

3.1 全体の構成

図 2 にエッジサーバにおけるストリーム処理アーキテクチャのブロック図を示す．ストリーム処理アーキテクチャは，センサデータを受信し各処理部への入力を行うデータ入力部と，統計データを作成する統計処理部，センサデータを保存する保存処理部，クラウドサーバに統計データを送信する MQTT クライアント部から構成される．

データ入力部のセンサデータ受信部は，制御装置から送信されるセンサデータを受信し，統計処理部と保存処理部双方にデータを送信する．統計処理部は，クラウドサーバへ送信するためのデータを作成し，保存処理部は制御装置からのセンサデータを全てドキュメント DB に保存する．MQTT クライアント部は統計処理部からデータを受け取り，クラウドサーバに対して統計データを送信する． 統計処理部で行う統計処理の内容は，統計処理ルールに規定され予め継続クエリ処理部に通知される．

データ入力部では，受信したセンサデータに，統計処理と保存処理に用いる識別情報を付与する．識別情報はセクション番号，シーケンス番号，センサ番号の 3 種類から構成される．セクション番号は，統計処理の間隔ごとにイン

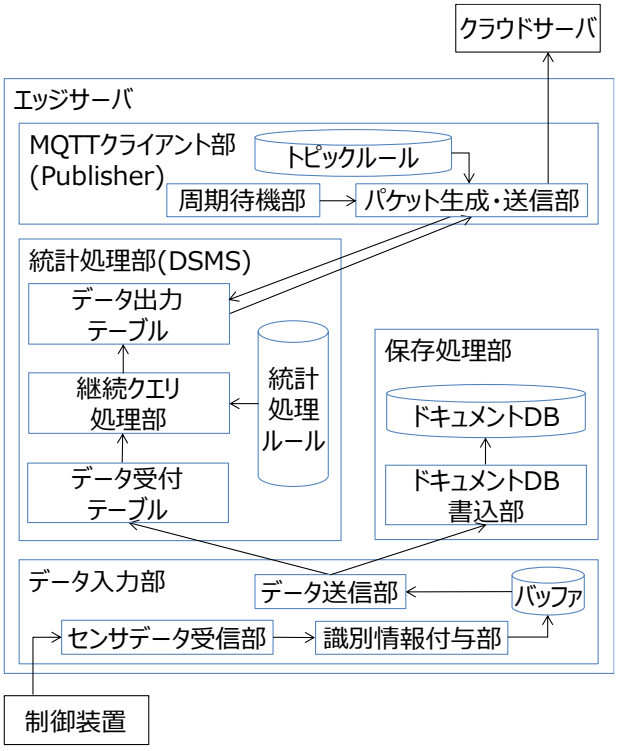

図 2　ストリーム処理アーキテクチャ

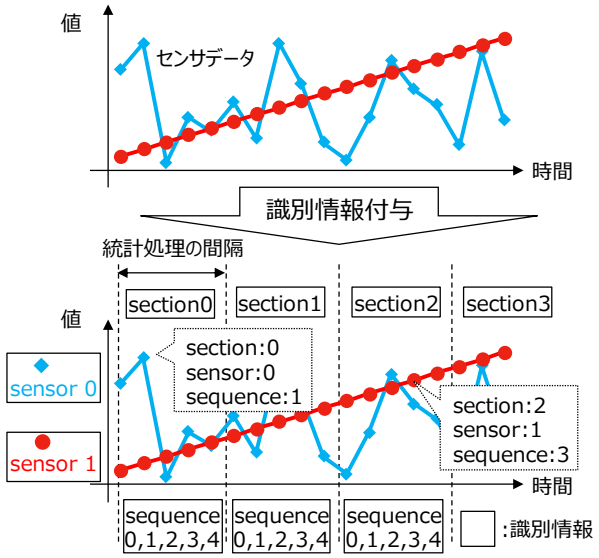

図 3　センサデータと識別情報の関係

クリメントする番号であり，統計データとドキュメント DB に保存したセンサデータを紐づけるために用いられる．シーケンス番号は，センサデータを受信するごとにインクリメントする番号であり，セクション内のデータを区別するために用いられる．センサ番号は，センサごとに一意に設定する識別子であり，複数のセンサを区別するために事前に設定される．

図 3 にセンサデータと識別情報の関係を示す．統計処理の間隔ごとにセクション番号が，セクション内の時系列順にシーケンス番号が，センサごとにセンサ番号が割り振られる． これらの識別情報とセンサデータはバッファに一時保存され，セクション単位で統計処理部と保存処理部に入力されることで統計処理や保存処理にかかる負荷を軽減することができる．

統計処理部と保存処理部で行う処理の詳細については，それぞれ 3.2 節，3.3 節で述べる．

MQTT クライアント部ではデータ出力テーブルから統計データを一定周期で取得し，クラウドサーバへ MQTT パケットを送信する．MQTT パケット送信の際に，センサ番号から一意に定まるトピックを統計データに付与することで，クラウドサーバでのデータ分類と保存が容易となる．

3.2 統計処理

ストリーム処理アーキテクチャではエッジサーバにおける統計処理のリアルタイム化のためデータストリーム管理システム(DSMS：Data Stream Management System)を適用する． DSMS は，センサデータにタイムスタンプを付加したデータのまとまりを構成し，事前に指定した問い合わせを繰り返し適用し続ける継続クエリ処理によって入力値に応じた出力をリアルタイムに行うシステムである[4], [5]. DSMS は，ウインドウ機能によって直近の一定期間のセンサデータのみを入力とし処理することができる．社会インフラ向けには，高度道路交通システムや BEMS (Building Energy Management System)などへの DSMS の適用が検討されている[6], [7].

ストリーム処理アーキテクチャの統計処理部に DSMS を適用し，センサデータに対して統計処理を行う流れについて説明する．データ受付テーブルには，センサデータと識別情報を合わせたデータのまとまりが入力される．継続クエリ処理部において，統計処理ルールから予め通知された統計処理の内容に従って，データ受付テーブルに入力されたデータのまとまりに対して統計処理が行われる．継続クエリ処理部で処理された統計処理の結果は逐次，データ出力テーブルに上書きされる．

図 4 に統計処理部でのデータ受付テーブルとデータ出力テーブルの構成と，データ受付テーブルからセンサデータの平均値を算出して統計データを生成する例を示す．データ受付テーブルでは，識別情報であるセクション番号，センサ番号，シーケンス番号とセンサデータ(data)を構成要素としている．data の添字は 1 から M までで，M はデータ受付テーブルの要素数とする．また，データ出力テーブルでは，識別情報であるセクション番号，センサ番号とセンサデータから算出された統計データ(statistic)を構成要素としている．statistic の添字は 1 から N までで，N はデータ出力テーブルの統計結果出力数とする．セクション番号，センサ番号が同じでシーケンス番号が異なるデータのまとまりに対して統計処理を行うため，シーケンス番号はデータ出力テーブルの構成要素には含まれない．統計処理の内容として，GROUP BY 句によってセクション番号とセンサ番号が同じ組み合わせとなるセンサデータを集計し，data から平均値を算出し statistic の値とする．data と statistic の添字の関係は，それぞれ同じ添字が対応し $M = N$ としている．図 4 の例では，データ受付テーブルに 4 行のデータのまとまりが入力されている．ここで，入力された 1 行のデータのまとまりをデータ行とする．入力されたデータ行の内，

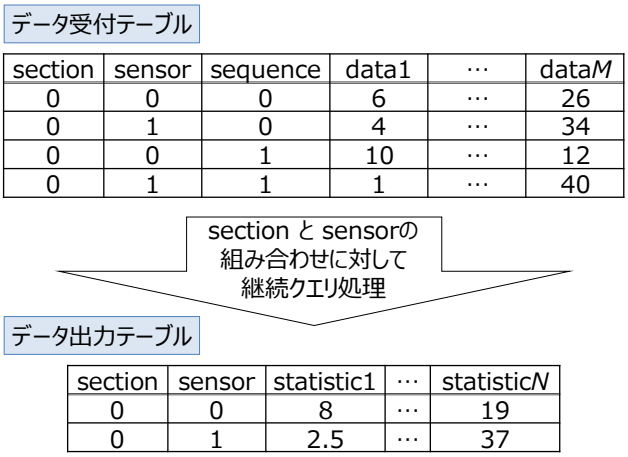

図 4　DSMS における統計データ生成

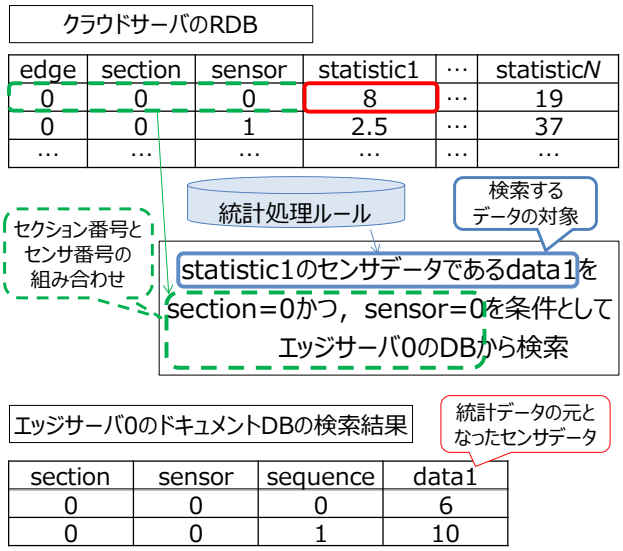

図 5　各 DB におけるデータの対応

セクション番号とセンサ番号がともに 0 である data1 は 6 と 10 であることから，その平均値は 8 となりデータ出力テーブルの(セクション番号，センサ番号，statistic1)が(0,0,8)として更新される．さらにここからデータ受付テーブルに(セクション番号，センサ番号，シーケンス番号，data1)が(0,0,2,5)となるデータ行が入力された場合は，データ行の data1 の 5 とデータ出力テーブルの statistic1 の 8 から平均値 7 が算出され，データ出力テーブルの該当箇所が 7 に更新される．

統計処理部では処理を高速化するため，メモリ上で継続クエリ処理が行われる．そのため，*M* と *N* が増えると，統計処理部で必要になるメモリ量が増える．また *M* と *N* が増加すると，データ受付テーブルに入力されるデータ行の数が減るため統計処理部で処理するクエリ数が減る．*M* と *N* は，処理の目標時間とエッジサーバのメモリ量に応じて適切に設定する必要がある．

3.3 保存処理とクラウドサーバとの連携

統計処理部では入力されたデータ行を保存せずに処理を行うため，データ保存用途には 2 章で述べたドキュメント DB を適用する．

エッジサーバ内のセンサデータとクラウドサーバ内の統計データを紐づけるために，保存処理部では 3.1 節で述べた識別情報であるセクション番号，シーケンス番号，センサ番号とともにセンサデータを保存する．保存処理部では，統計処理部と並行して処理が実行される．

データ利活用時におけるエッジサーバの保存処理部とクラウドサーバの RDB との連携動作について述べる．RDB ではセクション番号とセンサ番号，統計データの他にエッジサーバを区別するエッジ番号を付与して保存する．センサデータと統計データの相関を確認するため，クラウドサーバはエッジサーバが持つ統計処理ルールを保持する．ドキュメント DB と RDB のデータを対応させ，クラウドサーバが統計データの元となったセンサデータを検索する例を図 5 により説明する．なお，本例における統計処理の内容は図 4 で示した例と同一としている．クラウドサーバは statistic1 の元となったセンサデータを検索する際に，自身の保持する統計処理ルールから statistic1 の元となったセンサデータが data1 より作成されることを把握し，検索するデータの対象を data1 とする．クラウドサーバは data1 に関連する RDB のエッジ番号からエッジサーバを識別し，RDB のセクション番号とセンサ番号の組み合わせが同じセンサデータを検索することで，統計データの元となったセンサデータをエッジサーバから取得することができる．

4. 評価

4.1 試作評価

ストリーム処理アーキテクチャにおける統計処理および保存処理に関するリアルタイム性の検証として，本アーキテクチャを PC 上にソフトウェア実装し，統計処理部と保存処理部でのデータ格納への所要時間および，所要メモリ量を評価する．統計処理と保存処理で扱うデータには，大規模プラントを想定して 120,000 点の 4byte データを用意する．統計処理の内容として，入力されたデータに対して平均値を算出する図 4 と同様の処理を設定する．評価に用いた PC の仕様を表 1 に示す．統計処理部における DSMS には PipelineDB[8]，保存処理部におけるドキュメント DB には MongoDB[9]を使用する．

4.2 評価方法および結果

統計処理部と保存処理部でのデータ格納への所要時間を評価するため，データ受付テーブルとドキュメント DB への insert 処理に要した時間を計測した．なお，それぞれの処理の目標時間は 500ms 以内を想定している．

統計処理における評価として，PipelineDB に構築したデータ受付テーブルへの insert 処理に要した時間を計測した．

表 1　機器仕様

項目	仕様
OS	Ubuntu 16.04 LTS
CPU	2core 1.5GHz
Memory	4GB
PipelineDB version	0.9.8
MongoDB version	3.6.5

データ受付テーブルに対して 1 度に insert するデータ行の総数を L として，L と 3.2 節で述べたデータ受付テーブルの要素数 M と，データ出力テーブルの統計結果出力数 N を変化させたときの insert 処理に要した時間を表 2 に示す．(L, M, N) = (100, 100, 100)の条件で所要時間が最短の 0.216s となった．また，表 2 の条件における PipelineDB の所要メモリ量を表 3 に示す．(L, M, N) = (100, 100, 100)の条件でのメモリ量は 443,608KB となった．

保存処理における評価として，MongoDB への insert 処理に要した時間を計測した．ドキュメント DB に対して 1 度に insert するデータ行の総数を 1 として，ドキュメントフィールドの要素数を変化させたときの insert 処理に要した時間を表 3 に示す．なお，ドキュメント DB に対して 1 度に insert するデータ行の総数は L に相当し，ドキュメントフィールドの要素数は M に相当し，保存処理には出力がないため N は存在しない．要素数が 1000 となる条件で所要時間が最短の 0.085s となった．また，表 3 の条件における MongoDB の所要メモリ量を計測したところ 96,456KB となった．

PipelineDB と MongoDB の L と M が一致した L = 1, M = 100 の条件において，所要時間が双方 500ms 以内となり，目標時間を満足することを確認した．

5. 考察

DB 構成と所要時間とメモリ量の関係について考察する．

表 2　PipelineDB における insert の所要時間

L	M	N	所要時間[s]
100	1	1	1.952
1000	1	1	0.658
10000	1	1	1.207
10	10	10	1.677
100	10	10	0.325
1000	10	10	0.285
1	100	100	1.834
10	100	100	0.324
100	100	100	0.216
1	200	200	1.204
10	200	200	0.273
100	200	200	0.357

表 3　PipelineDB の所要メモリ量

M	N	メモリ量[KB]
1	1	14,556
10	10	108,140
100	100	443,608
200	200	1,655,152

表 4　MongoDB における insert の所要時間

要素数	所要時間[s]
10	1.361
100	0.192
1000	0.085

表 2 より M = N とするとデータ受付テーブルへの insert 処理数は 120,000／(L×M)となり，L,M を増加させることで insert 処理数が少なくなる．ここで，継続クエリ処理においては L×M が大きくなることで継続クエリ処理の計算量が少なくなり，所要時間が短くなると予想できる．一方で，(L, M, N) = (1000, 1, 1)と(L, M, N) = (10000, 1, 1)を比較すると，L×M の増加に反して所要時間が長くなっている．これは 1 つの insert 処理が長くなり構文解析に時間がかかることが原因と考えられる．

表 4 のフィールドの要素数と所要時間の関係についても表 2 の結果と同様に insert 処理数の減少による所要時間の減少が確認できる．

PipelineDB と MongoDB への insert 処理数が同条件となる表 2 における(L, M, N) = (1, 100, 100)のときと，表 4 における要素数 100 のときを比較すると，PipelineDB では所要時間が 1.834s であるのに対して MongoDB では 0.192s と，PipelineDB の方が MongoDB よりも insert に要した時間が長い．さらに，PipelineDB では(M, N) = (100, 100)のときのメモリ量が 443MB であるのに対して，MongoDB では 96MB となりメモリ量も PipelineDB の方が多くなり，PipelineDB がボトルネックとなる結果となった．評価に用いた試作を実際の制御システムへ適用する際は，統計処理の目標時間とメモリ量を考慮して PipelineDB の設計をする必要がある．

6. おわりに

本稿では，データ分散配置システムにおける，常時生成されるセンサデータに対し統計処理と保存処理をリアルタイムに行うストリーム処理アーキテクチャを提案した．また，統計処理と保存処理に要する時間を評価し，双方とも目標時間を満足する結果が得られリアルタイムに処理できることを確認した．今後は，継続してデータを格納したときの所要時間の評価を行うと共に，実際の制御システムへの適用に向けての課題抽出や DSMS とドキュメント DB における要素数の設定に取り組んでいく．

参考文献

[1] 原辰次, 本多敏: “超スマート社会におけるシステム科学技術概論”, 計測と制御, Vol. 55, No. 4, pp. 284-287 (2016)
[2] 高仲徹, 大西直哉, 金井遵: “広域監視制御システム向けデータ分散配置アーキテクチャの検討”, 電子情報通信学会ソサイエティ大会, B-7-37 (2017)
[3] 松山拓紀, 金井遵, 大西直哉, 高仲徹: “データ分散配置アーキテクチャにおけるリアルタイムデータ格納方式の検討”, 電気学会全国大会, 3-095 (2018)
[4] Abadi, D., Carney, D., Cetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N. and Zdonik, S. “Aurora: A New Model and Architecture for Data Stream Management”, VLDB Journal, Vol. 12, No. 2, pp. 120-139 (2003).
[5] 赤間浩樹, 内山寛之, 西岡秀一, 内藤一兵衛, 谷口展郎, 長谷川知洋, 兵藤正樹, 三浦史光, 山室雅司, 櫻井紀彦: “追記・参照型データ管理システムの設計と評価”, 情報処理学会論文誌, Vol. 49, No. 2, pp. 749-764 (2008)
[6] 渡辺陽介, 高木建太朗, 手嶋茂晴, 二宮芳樹, 佐藤健哉, 高田広章: “協調型運転支援のための交通社会ダイナミックマップの提案”, データ工学と情報マネジメントに関するフォーラム (2015)
[7] D. Anjos, P. Carreira, and A. P. Francisco. “Real-Time Monitoring of Building Energy Metering Networks”, INESC-ID Technical Report, December 2015.
[8] PipelineDB https://www.pipelinedb.com/
[9] MongoDB https://www.mongodb.com/