MapReduce処理系SSSにおけるContinuous MapReduceの実装

中田 秀基[1]　小川 宏高[1]　工藤 知宏[1]

概要：継続的に生成されるストリームデータを、大容量データと付きあわせて処理するためのシステムを提案する。従来のストリーム計算システムは、オンメモリの比較的少量のデータしか参照できない。一方Hadoopに代表されるファイルベースのMapReduceシステムを拡張し、ストリームデータに対応させる研究も存在するが、ストリームデータとストレージ上のデータをうまく組み合わせて処理することのできる処理系はない。我々の提案するSSSは、間欠的に起動するMapper / Reducerプロセスでストリームデータの処理を行う。この際にストレージ上の既存データとのマージ操作を行うことで、ストレージ上の大容量データとストリームデータを付きあわせて処理することが可能になる。本稿ではSSSのストリームデータ処理機構の概要と、ストリームデータ処理の予備的評価の結果を示す。予備評価の結果、提案システムは1ノードあたりおよそ秒間0.14Miレコードを処理できることが確認できたが、読み出しスレッドとの干渉により間欠的に書き込みスループットが大幅に低下し、平均としては秒間0.095Miレコード程度となることがわかった。

Continuous MapReduce implementation with SSS-MapReduce

Hidemoto Nakada[1]　Hirotaka Ogawa[1]　Tomohiro Kudoh[1]

Abstract: We propose a MapReduce based stream processing system, called SSS, which is capable of processing stream along with large scale static data. Unlike the existing stream processing systems that can work only on the relatively small on-memory data-set, SSS can process incoming streamed data consulting the stored data. SSS processes streamed data with continuous Mappers and Reducers, that are periodically invoked by the system. It also supports merge operation on two set of data, which enables stream data processing with large static data. This paper describes overview of the stream processing with SSS and shows preliminary evaluation results. The results showed that, on the proposed system, one node can process 0.14Mi records/sec at peak, but in average the throughput goes down to 0.095Mi records/sec, because of the interfere from the Mapper that is periodically invoked by the system.

1. はじめに

カメラなどのセンサーデータや各種Webサービス類のログなどの、継続的に生成されるデータのストリームを処理する枠組みへの要求が増大している。このようなデータを処理する枠組みとしては、ストリーム計算システム [1] [2] が提案されている。これらのシステムは、データ処理のレイテンシを非常に重視するため、オンメモリの比較的少量のデータのみを参照して動作する場合が多い。この構造はアルゴリズムトレードのような特殊なアプリケーションに対しては有効だが、応用範囲は限定されている。

我々は、ストリームデータから十分な情報を引き出すためにはオンメモリのデータだけではなく、ストレージ上のより大容量のデータを参照した処理が必要だと考える (図1)。このために、本稿では、継続的に生成されるストリームデータを、大容量データと付きあわせて処理することのできるMapReduceシステムSSSを提案する。

SSSのターゲットはオンライントレードのような低レイテンシを要求されるストリーム処理ではなく、数分から数

[1] 独立行政法人 産業技術総合研究所
National Institute of Advanced Industrial Science and Technology

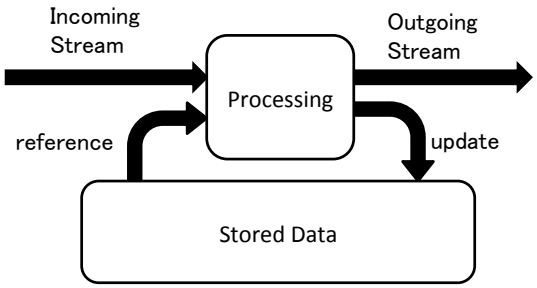


図 1 Streaming Computation with Bigdata.

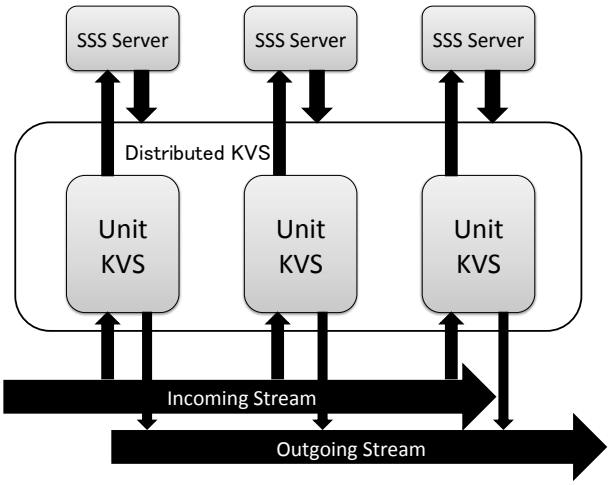


図 2 Overvieew of SSS.

時間程度のレイテンシが許されるが、大容量の静的データを参照する必要があるアプリケーションである。このようなアプリケーションとしては、ログ監視を行いアラートを上げるシステムや、センサーデータを統合し過去のデータと参照して変化を検出するシステムなどが考えられる。

我々の提案する SSS [3] は、KVS(Key Value Store) をベースとした MapReduce 処理を行うシステムである。間欠的に起動する Mapper / Reducer プロセスでストリームデータの処理を行う。また、MergeReducer と呼ばれる機構を用いることで、ストリームからのデータとストレージ上の既存データとのマージ操作を行い、我々の提案する SSS [3] は、間欠的に起動する Mapper / Reducer プロセスでストリームデータの処理を行う。また、MergeReducer と呼ばれる機構を用いることで、ストリームからのデータとストレージ上の大容量データとのマージ操作を行うことで、大容量データを参照したストリーム操作が実現できる。

本稿の構成は以下のとおりである。2 節で SSS の概要と実装について述べる。3 節で SSS によるストリーム処理の実装について述べる。4 節に予備的評価の結果を示す。6 節に関連研究をまとめる。7 節で結論と課題について延べる。

2. SSS の概要

SSS[3][4] は、われわれが開発中の MapReduce 処理系である。SSS は HDFS のようなファイルシステムを基盤とせず、分散 KVS を基盤とする点に特徴がある。入力データは予めキーとバリューの形で分散 KVS にアップロードしておき、出力結果も分散 KVS からダウンロードする形となる。

SSS ではデータをキーに対するハッシュで分散した上で、Owner Compute ルールにしたがって計算を行う。つまり、各ノード上の Mapper/Reducer は自ノード内のキーバリューペアのみを対象として処理を行う。これは、データ転送の時間を削減するとともに、ネットワークの衝突を防ぐためである。

Mapper は、生成したキーバリューペアを、そのキーでハッシングして、保持担当ノードを決定し直接書き込む。書きこまれたデータは各ノード上の KVS によって自動的にキー毎にグループ分けされる。これをそのまま利用して、Reduce を行う。つまり SSS においては、シャッフルは、キーのハッシングと KVS によるキー毎のグループ分けで実現されることになる。

SSS のもうひとつの特徴は、Map と Reduce を自由に組み合わせた繰り返し計算が容易にできることである。前述のように、SSS では Map と Reduce の間でやりとりされるデータも KVS に蓄積されるため、Map と Reduce が 1 対 1 に対応している必要がない。したがって、任意個数、段数の Map と Reduce から構成される、より柔軟なデータフロー構造を対象とすることができる。

2.1 SSS の構成

SSS の構成を図 2 に示す。各ノード上では、SSS サーバと単体 KVS のサーバが稼働する。SSS サーバはユーザの書いた Mapper や Reducer を起動し、KVS から読みだしたデータをフィードする役割を果たす。

2.2 SSS の分散 KVS 実装

SSS は、単体 KVS をキーに対するハッシングで分散化したものを分散 KVS として用いる。図 2 に示したとおり、単体 KVS は独立して動作しており、相互の通信は行わない。KVS に対するクライアントである SSS サーバ群が共通のハッシュ関数を利用することで、総体としての分散 KVS が構成されている。

単体 KVS としては、Tokyo Cabinet[5] を、SSS が多用するソート済みデータのバルク書き込み、およびレンジに対するバルク読み出しに特化してカスタマイズしたもの [6] を用いた。Tokyo Cabinet へのリモートアクセスには、C++で実装された独自の通信レイヤ [7] を用いている。

2.3 タプルグループ

SSS ではデータの空間を複数の名前空間に分割して利用する。この名前空間をタプルグループと呼ぶ。Mapper や Reducer は、タプルグループからタプル（キーとバリュー

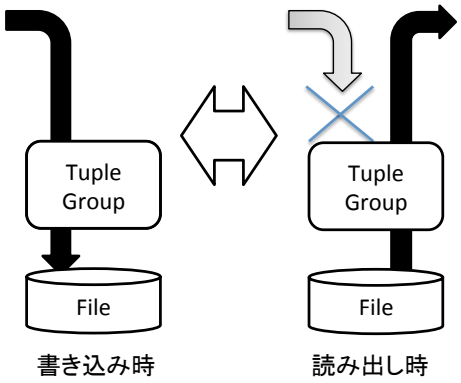


図 3 読み出し時の書き込みブロック

のペア）を読み出し、他のタプルグループにタプルを書き出す。

3. SSS によるストリーム処理

3.1 ストリーム入出力

SSS では、入力ストリームは特定のタプルグループへ継続的な書き込みとして表現される。このタプルグループは入力バッファとして機能する。同様に出力は特定のタプルグループからの継続的な読み出しとして実現される。

3.2 間欠的 Mapper / Reducer

SSS におけるストリーム処理は Mapper/Reduer を継続的かつ間欠的に実行することで実現する。間欠実行の頻度はユーザが設定する。間欠的 Mapper/Reducer は起動のたびに、そのタイミングで入力タプルグループに存在するデータをすべて読み出し、消去する。

この際、競合を避けるために、読み出し消去を行うスレッドはタプルグループに対してロックをかける。書き込みスレッドは、ロックが開放されるのを待つため、一時的に書き込みができなくなる (図 3)。後述するように、この挙動は性能上問題となっており、今後改良を行う予定である。

3.3 MergeReducer

MergeReducer は、複数のタプルグループからの入力を処理する特殊な Reducer である。MergeReducer は merge sort のように機能する。タプルグループの一方を入力ストリームのバッファとすることによって大容量データとストリームデータの双方を参照したアルゴリズムを容易に記述することができる。

図 4 に、この様子を示す。MergerReducer の入力の一方を入力ストリームからのバッファとなるタプルグループとし、もう一方を静的な大容量データのタプルグループとする。この MergeReducer を間欠的に繰り返し起動する。起動する毎に MergeReducer は、双方の入力タプルスペースの内容をキーでマージしながら読み出し、ユーザが定義したメソッドに引き渡す。この動作では、静的データを間欠

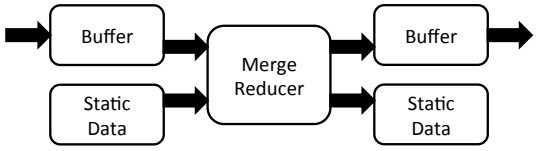


図 4 MergeReducer の挙動

起動のたびに全て読みだすことになるので、間欠起動の頻度を十分小さくしておくことが重要である。

下に、MergeReducer の merge メソッドのプロトタイプを示す。

```
void merge(Context         context,
           T0              key,
           Iterable<T1>    values0,
           Iterable<T2>    values1,
           Output<T3, T4>  output);
```

4. 予備評価

SSS のデータストリーム処理性能を確認するために、単体ノードでのデータ処理のテストを行った。入力データは Apache Web Server のログを模したものである。これを 10000 レコードを 1 単位として SSS に対して書き込む。書き込み単位間には 10ms のインターバルを設けた。1 レコードのサイズは 300 バイト程度である。

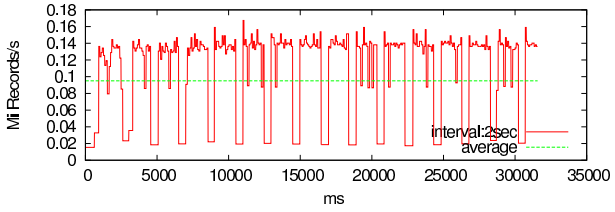
実験は、Intel(R) Xeon(R) W5590 3.33GHz のサーバを用いた。ストレージには、 Fusion-io ioDrive Duo 320GB を用いている。

Continuous Map 処理の起動インターバルを 2 秒、5 秒、10 秒として実行した。結果を図 5 に示す。いずれの場合も平常時には秒間 0.14Mi レコード程度のスループットとなっているが、Continuous Map が起動した際に、スループットが大きく低下している事がわかる。また、スループットが低下している時間は、起動インターバルの増大に伴って増大している。これは Map プロセスの読み出し時に、KVS に対してロックをかけ、書き込みを一時的に停止しているためである。起動インターバルが大きい場合にはそれだけデータが溜まっているため、ロックされる時間も長くなる。起動インターバルを小さくすると、少量のデータ処理ですむため、ロック時間は短くなるため、書き込み不能の時間を短くすることができる。平均スループットはいずれの場合も 0.095Mi 程度と変わらない。これは読み出しにかかる時間の総計はインターバルにかかわらず一定であるためで、妥当な結果である。

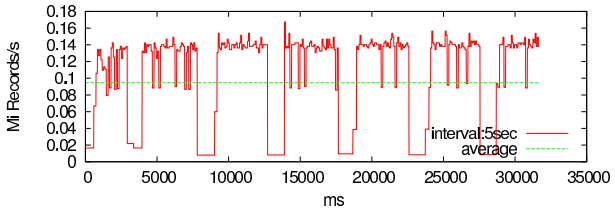
5. 議論

5.1 スライディング・ウィンドウ処理

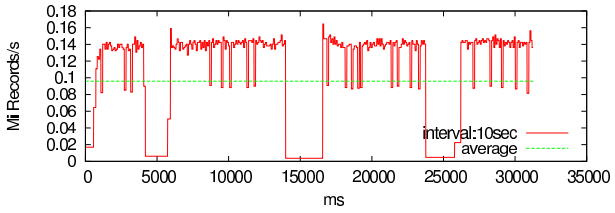
ストリーム処理においては、スライディングウィンドウと呼ばれる単位に対して処理を行う場合がある。これは、ウィンドウと呼ばれるタイムスパンに対する処理を、ウィン



読み出しインターバル 2 秒



読み出しインターバル 5 秒



読み出しインターバル 10 秒

図 5 ログレコード入力のスループット.

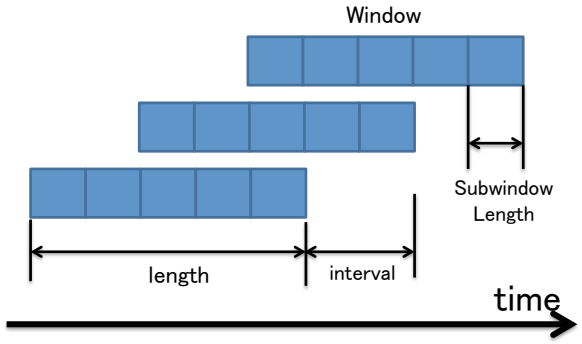


図 6 スライディング・ウィンドウ.

ドウをずらしながら行う処理である [8]。スライディング・ウィンドウは、長さ（*length*）とインターバル（*interval*）の 2 つの値で定義される（図 6）。

SSS は、スライディングウィンドウを直接サポートしないが、Reducer の後ろに、独立した別の Reducer（Post Reducer）を付け加えることでスライディングウィンドウを実現することができる。Mapper、Reducer、Post Reducer を起動するインターバルは、スライディングウィンドウのインターバルと長さの最少公倍数 (図 6 中の *subwindowLength*) に設定する。Post Reducer はキーに対応した少量のオンメモリリングバッファを持つ。リングバッファの長さは、*length*/*subwindowLength* とする。

例として、長さ 15 分、インターバル 6 分のスライディングウィンドウで移動平均を取る事を考える。サブウィンドウ長は 15 分と 6 分の最少公倍数である 3 分となる。Post Reducer の持つリングバッファ長は 5 となる。Mapper/Reducer は 3 分に一度起動し、3 分間のデータを集計して出力する。Post Reducer も 3 分に一度起動し、Reducer の出力を取り込み、リングバッファに格納する。そして、起動 2 回につき 1 回だけリングバッファの内容から平均値を参照して出力を行う。図 7 にこの様子を示す。

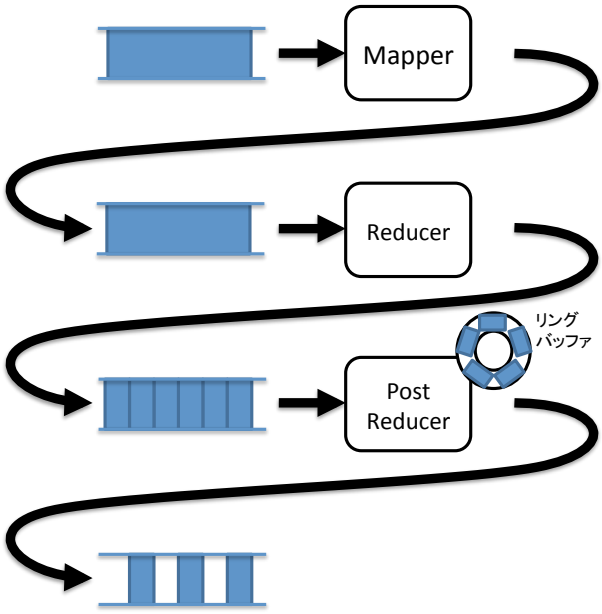


図 7 スライディング・ウィンドウの処理

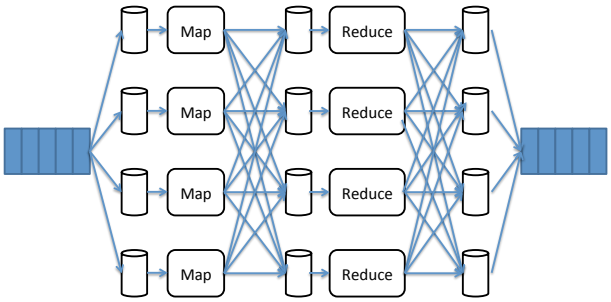


図 8 SSS におけるデータフロー

5.2 処理順序の保証

SSS ではストリームイベントに対する処理順序を厳密に保証することはできない。図 8 に SSS におけるデータフローを示す。SSS では、入力されたストリームは複数のワーカノードに分散されて処理される。さらに個々のワーカノードの中にも複数の処理スレッドが並列に実行され、ストリームイベントはスレッド間に分散される。このため、ノードからの出力は Reducer に渡される際にシャッフルされるが、各ノード、各スレッドでの処理速度が一定ではないため、Reducer に渡る際にイベントの順序を保証することはできない。

このような特性は、厳密なストリーム処理においては問題になる可能性がある。アプリケーションの特性に照らして検討をすすめる必要がある。

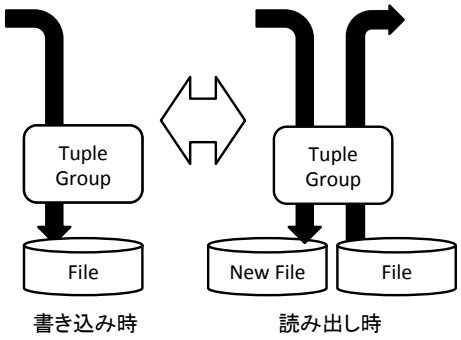


図 9 ファイルローテーションによるブロッキングの解消

5.3 読み出しによる書き込みブロックの排除

Mapper が読み出し時に書き込み性能が停止するのは、読み出し側のスレッドが、レースコンディションを避けるために、タプルグループに対してロックをかけているためである。Mapper 起動のインターバルを小さく設定してロック時間を短くすることで影響を小さくすることは可能だが、書き込み側にバッファが十分にない場合には、データの取りこぼしにつながる。

安定した書き込みのために、読み出し時にデータベースファイルをローテーションすることを計画中である。この場合は、読み出しは古いファイルに対して行われるのに対して、書き込みは新たなファイルに行われるため、ロックをかける必要がなくなる。

6. 関連研究

6.1 Hadoop Online Prototype

HOP(Hadoop Online Prototype)[9] は、Hadoop を改変し、Mapper と Reducer, さらには Reducer と次のイタレーションの Mapper を直接ソケットで接続し、パイプライン的にデータを流すことで、繰り返し処理の高速化を狙った処理系であるが、同時に Continous Mapper/Reducer を用いた Continuous query もサポートしている。

HOP は Hadoop をベースにして入るが、Continuous query を実行する際には、Mapper も Reducer も入力をストリームとして受け取って動作する。このため、大容量データを参照した動作はできない。

6.2 C-MR(Continuous MapReduce)

C-MR [8] は、単一ノード上の多数のスレッドで動作するストリームプロセッシングシステムである。C-MR は本稿同様に、Sliding Window 処理機能を持つが、SSS のそれよりもはるかに厳密な Windowing を行うことができる。

一方で単一ノード上でしか動作しないため、入力ストリームの増大に対して、ノード数を増やしてスケールアウトする戦略を取ることはできない。

6.3 S4

S4[10][11] は、Yahoo が開発し Apache に寄贈したストリーム処理プログラミングの分散フレームワークである。S4 の計算モデルはデータフロー計算と類似している。SSS で計算対象となるデータは、イベントと呼ばれ、キーとバリューで表現される。演算は PE（Processing Element）と呼ばれるユニットで行われる。イベントはキーで識別され対応する PE に送られる。対応する PE のインスタンスがない場合には自動的に生成される。PE は受け取ったイベントを処理して、新しいイベントを作成する。この時、複数のイベントのマージ処理を行うこともできる。

6.4 DEDUCE

Deduce は IBM のストリーム処理システム System S に対して MapReduce のサポートを追加したものである。System S のワークフローの一部に MapReduce API で記述されたジョブを組み込む事ができる。また、静的なファイルを読みだして処理することもできる。

DEDUCE のモティベーションは、大容量データを用いたストリーム処理ということでわれわれのそれに近いが、計算の構成としては大きく異る。DEDUCE ではメインのストリーム処理は通常のストリーム処理として行い、その入力の一分として、バッチ的に行った MapReduce の結果を取り込む構成となっている。

6.5 iMR (in-situ MapReduce)

iMR[12] は、大容量のログ処理に特化した MapReduce システムである。大容量ログ処理を継続的に行うために、ログを出力するサーバ群の内部で Continuous MapReduce を行い、その結果のみを外部のファイルシステムに格納するアーキテクチャである。

7. おわりに

KVS を基盤とした MapReduce 処理系 SSS によるストリーム処理の概要を示した。SSS では間欠的に動作する Mapper と Reducer によってストリーム処理を実現する。間欠的 Mapper・Reducer は読み出しタプルグループからデータを読み出しつつ消去を行う。

本稿では既存の SSS 処理系にわずかな改変を施すことでストリーム処理を実現できることを示した。これは、KVS を基盤とする MapReduce 処理系の柔軟性を示すものである。また、MergeReducer のように複数の入力を同時に処理する操作が容易に実現できたのも KVS を基盤としているためである。

単体ノードを用いた予備評価の結果、平均して毎秒およそ 0.095Mi レコードの処理が可能であることが確認できた。また、読み出しのインターバルによっては長時間の書き込み不能時間ができてしまうことも分かった。

今後の課題としては、書き込みと読み込みのファイルを分離することで、読み出し時のロックを回避する方法の採用が挙げられる。また、書き込みそのもののスループットも、本来期待できるスループットと比較すると小さい。書き込みのチューニングによる高速化も課題の一つである。

謝辞

本研究の一部は，独立行政法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務「グリーンネットワーク・システム技術研究開発プロジェクト（グリーン IT プロジェクト）」の成果を活用している．

参考文献

[1] Abadi, D. J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A. S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y. and Zdonik, S.: The Design of the Borealis Stream Processing Engine, *Proc. of 2nd Biennial Conference on Innovative Data Systems Research (CIDR'05)* (2005).

[2] Gedik, B., Andrade, H., Wu, K.-L., Yu, P. S. and Doo, M.: SPADE: the system s declarative stream processing engine, *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD '08, New York, NY, USA, ACM, pp. 1123–1134 (2008).

[3] Ogawa, H., Nakada, H., Takano, R. and Kudoh, T.: SSS: An Implementation of Key-value Store based MapReduce Framework, *Proceedings of 2nd IEEE International Conference on Cloud Computing Technology and Science (Accepted as a paper for First International Workshop on Theory and Practice of MapReduce (MAPRED'2010))*, pp. 754–761 (2010).

[4] 中田秀基，小川宏高，工藤知宏：分散 KVS に基づく MapReduce 処理系 SSS，インターネットコンファレンス 2011 (IC2011) 論文集，pp. 21–29 (2011).

[5] FAL Labs: Tokyo Cabinet: a modern implementation of DBM, `http://fallabs.com/tokyocabinet/index.html`.

[6] 中田秀基, 小川宏高, 工藤知宏：MapReduce 処理系 SSS に向けた KVS の改良, 信学技報, Vol.112, No.2 CPSY2012-1 - CPSY2012-8，pp. 19–24 (2012).

[7] 中田秀基，小川宏高，工藤知宏：MapReduce 処理系 SSS における Key Value Store アクセス手法の改良，信学技報, Vol.112, No.173 CPSY2012-9 - CPSY2012-30，pp. 103–108 (2012).

[8] Backman, N., Pattabiraman, K., Fonseca, R. and Cetintemel, U.: C-MR: Continuously Executing MapReduce Workflows on Multi-core Processors, *Proceedings of the 3rd International Workshop on MapReduce and its Applications, MapReduce '12* (2012).

[9] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Elmeleegy, K. and Sears, R.: MapReduce online, *Proceedings of the 7th USENIX conference on Networked systems design and implementation*, NSDI'10, Berkeley, CA, USA, USENIX Association, pp. 21–21 (2010).

[10] : S4 distributed stream computing platform, `http://incubator.apache.org/s4/`.

[11] Neumeyer, L., Robbins, B., Nair, A. and Kesari, A.: S4: Distributed Stream Computing Platform, *Proceedings of International Workshop on Knowledge Discovery Using Cloud and Distributed Computing Platforms (KDCloud 2010)* (2010).

[12] Logothetis, D., Trezzo, C., Webb, K. C. and Yocum, K.: In-situ MapReduce for log processing, *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, HotCloud'11, Berkeley, CA, USA, USENIX Association, pp. 26–26 (2011).