Vol.2012-HPC-135 No.25
2012/8/2

MapReduce処理系SSSのアプリケーションによる消費電力測定

小川 宏高[1]　中田 秀基[1]　工藤 知宏[1]

概要：MapReduce は，大規模なデータインテンシブ計算の実装手段として広範に用いられている．その単純なプログラミングモデルゆえ，MapReduce はデータインテンシブ計算のみならず，より一般的な HPC 分野の並列分散アプリケーションのためのプログラミングツールとしても利用されつつある．われわれはイタレーションが高速で柔軟なワークフローの構成が可能な大規模データ処理システムの実現を目的として，MapReduce 処理系 SSS を開発している．Hadoop が HDFS と呼ばれる一種の分散ファイルシステムを基盤としているのに対して，SSS は分散キーバリューストアを基盤としている点が大きく異なる．一方で，こうした処理系の利活用によってますます大規模な計算リソースを必要とするようになると，ランニングコスト，とりわけ電力コストの問題は避けられない問題となる．われわれは消費電力に対して処理系をチューニングする必要があると考え，そのために実験用クラスタの消費電力を秒単位で計測・報告するシステムを構築した．その上で，本稿では English Wikipedia の全テキストデータを対象に Word Count を実行するベンチマーク実験を実施したので本稿ではその結果を示す．

Measuring Power Consumption on SSS MapReduce Framework by Using An Application Benchmark

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

Abstract: MapReduce has been very successful in implementing large-scale data-intensive applications. Because of its simple programming model, MapReduce has also been utilized as a programming tool for more general distributed and parallel applications, e.g., HPC applications. To realize a large-scale data processing system which supports faster iterations and more flexible workflows, we have been developing a new MapReduce framework called "SSS". Unlike Hadoop based on HDFS distributed file system, SSS is based on distributed key-value store (KVS). On the other hand, the more such systems are utilized, the more computing resources are needed. Hence, the running cost, especially the power-consumption cost, arises as an inevitable problem. We think we need to optimize our system to lessen the power consumption, and first we built a system which can visualize the power consumption data of our experimental cluster. And we performed the wordcount benchmarks using English Wikipedia text data. In this paper, we introduce our system and show the benchmark results.

1. はじめに

MapReduce[1] は，大規模なデータインテンシブアプリケーションの実装手段として知られ，実際にそのオープンソース実装である Hadoop[2] は広範に用いられている．その単純なプログラミングモデルゆえ，Hadoop はデータインテンシブアプリケーションのみならず，より一般的な HPC 分野の並列分散アプリケーションのためのプログラミングツールとしても利用されつつある．われわれはイタレーションが高速で柔軟なワークフローの構成が可能な大規模データ処理システムの実現を目的として，MapReduce 処理系 SSS[3], [4] を開発している．Hadoop が HDFS と呼ばれる一種の分散ファイルシステムを基盤としているのに対して，SSS は分散キーバリューストアを基盤としている

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

点が大きく異なる．これまでにいくつかのトイアプリケーションの他，合成ベンチマーク，K-means，PrefixSpan 法ならびに疎行列ベクトル積などで性能特性を明らかにし，多くの場合において Hadoop に対する性能面での優位性を実証してきた [5], [6], [7].

一方で，こうした処理系の利活用によってますます大規模な計算リソースを必要とするようになると，ランニングコスト，とりわけ電力コストの問題は避けられない問題となる．

電力コストを下げる，すなわち電力効率を高める一つの方法は，電力あたりの仕事量を増やすことである．単一ジョブの実行性能の改善（SSS を含め多数のエフォートがある）や，多数のバッチジョブやインタラクティブジョブを共有の計算リソースで実行することによる稼働率の向上は，電力効率の向上に寄与する．

もう一つの方法は，ワークロードあたりの消費電力量を減らすことである．近年，（人間を含めた）外界からの逐次的な入力・操作に応じて，データ構造を更新し続けるタイプのワークロードを MapReduce をイテラティブに実行することで実現する試みが増えつつある．例えば，大規模な SNS ではユーザの入力に応じてソーシャルグラフをメンテナンスし続ける処理が必要となる．また，サイバーフィジカルシステムで期待されているように，実世界から取得される逐次的な情報を元に機械学習等の処理をおこない，何らかの実世界へのフィードバックを返す処理も挙げられる．このような処理においては，ワークロードは持続的であり，負荷や頻度などは外界によって条件付けされる．したがって，実効性能の改善自体は有効だが，利用リソース量の事前予測が困難なため，稼働率の向上には限界がある．このようなケースでは，消費電力量自体を削減することが必要である．

このような観点から，われわれは消費電力に対して処理系をチューニングする必要があると考え，そのために実験用クラスタの消費電力を秒単位で計測・報告するシステムを構築した．その上で，本稿では English Wikipedia の全テキストデータを対象に Word Count を実行するベンチマーク実験を実施した．ストレージとして Fusion-io ioDrive Duo，Crucial RealSSD C30，Fujitsu SAS HDD，MapReduce 処理系として SSS，Hadoop を用い，比較・評価をおこなった．

2. MapReduce/Hadoop/SSS

2.1 MapReduce

MapReduce とは，入力キーバリューペアのリストを受け取り，出力キーバリューペアのリストを生成する分散計算モデルである．MapReduce の計算は，Map と Reduce という二つのユーザ定義関数からなる．これら 2 つの関数名は，Lisp の 2 つの高階関数からそれぞれ取られている．Map では大量の独立したデータに対する並列演算を，Reduce では Map の出力に対する集約演算をおこなう．

一般に，Map 関数は 1 個の入力キーバリューペアを取り，0 個以上の中間キーバリューペアを生成する．MapReduce のランタイムはこの中間キーバリューペアを中間キーごとにグルーピングし，Reduce 関数に引き渡す．このフェイズをシャッフルと呼ぶ．Reduce 関数は中間キーと，そのキーに関連付けられたバリューのリストを受け取り，0 個以上の結果キーバリューペアを出力する．各 Map 関数，Reduce 関数はそれぞれ独立しており，相互に依存関係がないため，同期なしに並列に実行することができ，分散環境での実行に適している．また，関数間の相互作用をプログラマが考慮する必要がないため，プログラミングも容易である．これは並列計算の記述において困難となる要素計算間の通信を，シャッフルで実現可能なパターンに限定することで実現されている．

2.2 Hadoop

Hadoop は，代表的なオープンソースの MapReduce 処理系である（**図 1**）．Hadoop では，入力データは HDFS 上のファイルとして用意される．Hadoop は，まず MapReduce ジョブへの入力をスプリットと呼ばれる固定長の断片に分割する．次に，スプリット内のレコードに対して map 関数を適用する．この処理をおこなうタスクを Map タスクと呼ぶ．Map タスクは各スプリットに関して並列に実行される．map 関数が出力した中間キーバリューデータは，partition 関数（キーのハッシュ関数など）によって，Reduce タスクの個数 R 個に分割され，各パーティションごとにキーについてソートされる．ソートされたパーティションはさらに combiner と呼ばれる集約関数によってコンパクションされ，最終的には Map タスクが動作するノードのローカルディスクに書き出される．

一方の Reduce 処理では，まず Map タスクを処理したノードに格納されている複数のソート済みパーティションをリモートコピーし，ソート順序を保証しながらマージする．（結果的に得られた）ソート済みの中間キーバリューデータの各キーごとに reduce 関数が呼び出され，その出力は HDFS 上のファイルとして書き出される．

2.3 SSS

SSS[3], [4], [6] は，われわれが開発中の MapReduce 処理系である（**図 2**）．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．入力データはあらかじ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．これは煩雑に思えるかもしれないが，Hadoop の場合でも同様に HDFS を計算時のみに用いる一時ストレージとして運用しているケースも多く，それほ

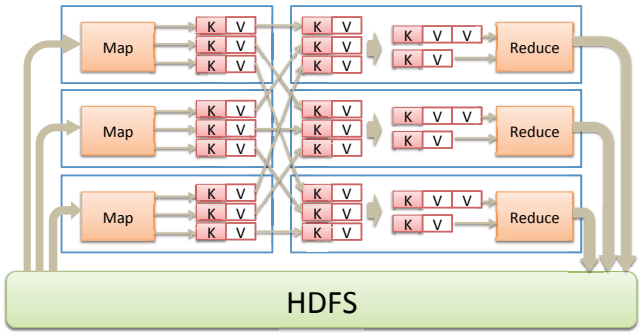

図 1 Hadoop でのデータの流れ

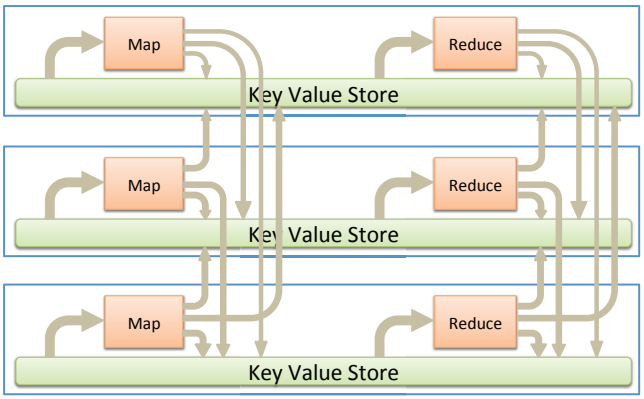

図 2 SSS でのデータの流れ

どデメリットであるとは考えていない．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 Computes ルールにしたがって計算をおこなう．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おこなう．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 をおこなう．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．

3. 消費電力計測実験

われわれは消費電力に対して処理系をチューニングする必要があると考え，そのために実験用クラスタの消費電力を秒単位で計測・報告するシステムを構築した．その上で，本稿では Wikipedia 英語版の全テキストデータを対象に Word Count を実行するベンチマーク実験を実施した．ベンチマークでは，中規模と小規模のデータセットを用い，ストレージとして Fusion-io ioDrive Duo，Crucial RealSSD C30，Fujitsu SAS HDD，MapReduce 処理系として SSS，Hadoop を用いて，比較・評価をおこなった．以下ではその詳細を述べる．

3.1 消費電力の計測方法

産総研では，安価な電力計測器を開発し，それを用いた電力可視化システムを所内の計算サーバー室に構築している [8], [9], [10].

図 3 に示すように電力計測器は計算機サーバー室内の分電盤からクランプ型電流センサーを用いて計測する．電力値は，センサー計測値を 1 秒間に 6400 回サンプリングしたデータから高精度に計算され，1 秒毎にデータ収集器に送られる．データ収集器は複数の電力計測器から集約したデータを 20 秒毎にまとめて Google App Engine で構築されたデータ収集サーバに保存する．保存されたデータは App Engine 上の可視化アプリケーションで閲覧できるほか，Web API を介して PC 上などから電流センサー単位・秒単位で取得することができる．

本稿の消費電力計測では，この Web API を用いて実験用クラスタの消費電力を連続的に取得して利用する．電力の測定はサーバ一台単位ではなく，実験用クラスタ全体となる．つまり，計算サーバの他にマスターサーバ，iSCSI ストレージサーバ，10Gbit Ethernet スイッチを含む．

参考までに，計算サーバ 16 ノードを電源オフにした状態で約 897W，ジョブを走らせていない状態で平均 3,793W の電力を消費している．

3.2 評価環境

評価には，**表 1** に示すように，1 台のマスターノードと 16 台のワーカーノード（ストレージノードと MapReduce 実行ノードを兼ねる）からなる小規模クラスタを用いた．各ノードは 10Gbit Ethernet で接続され，各ワーカーノードは Fusion-io ioDrive Duo 320GB（以降，ioDrive）と，Crucial RealSSD C30 256GB（以降，SATA SSD），Fujitsu MBA3147RC 147GB（以降，SAS SSD）を備えている．

使用している Hadoop のバージョンは，Cloudera Distribution for Hadoop 3 Update 4 である．dfs.replication を 1 に設定することで HDFS のレプリカ生成を抑制している．また，各 Map タスク，Reduce タスクが使用できるヒープのサイズは 4GB に設定してある．また，Reduce タスク数は 16 としている．

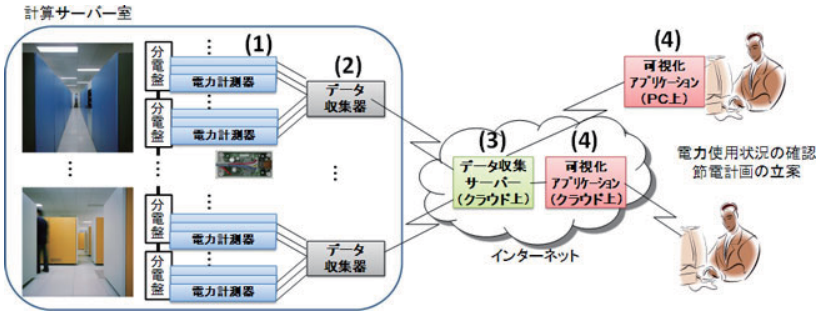

図 3 AIST 電力可視化システムの全体構成（[10] より引用）

表 1 Benchmarking Environment

# nodes	17 (*)
CPU	Intel(R) Xeon(R) W5590 3.33GHz
# CPU per node	2
# Cores per CPU	4
Memory per node	48GB
Operating System	CentOS 5.5 x86_64
Storage 1	Fusion-io ioDrive Duo 320GB
Storage 2	Crucial RealSSD C30
Storage 3	Fujitsu MBA3147RC 147GB/15000rpm
Network Interface	Mellanox ConnextX-II 10G Adapter
Network Switch	Cisco Nexus 5010

(*) one node is reserved for the master server.

3.3 ベンチマークアプリケーション

ベンチマークアプリケーションとしては Word Count を用い，対象データは English Wikipedia のデータベース [11] から抽出したテキストデータ約 7.8GB を利用する．このデータを元に以下の二つのデータセットを用意した．

- Regular
  全テキストデータ 7.8GB を 1MB ごとにスプリットしたもの
- Small
  テキストデータの先頭 10MB を 1KB ごとにスプリットしたもの

Hadoop においては，各スプリットを一個のファイルとして HDFS 上に，SSS においては，各スプリットを一個のキーバリューデータとして分散 KVS 上に，それぞれ格納して実行するものとする．Word Count 自体の実装は Hadoop では処理系に添付されているもの，SSS でも `http://sss.apgrid.org/packages` で配布されているパッケージに添付しているものを用いた．

ベンチマークでは，以下のシーケンスで実行している:

(1) 1 分間休止
(2) 入力データを HDFS，分散 KVS に保存
(3) ファイルシステムのページキャッシュ，dentry，inode を強制開放（`/sbin/sysctl -w vm.drop_caches=3`）
(4) SSS の場合には，Tokyo Tyrant の再起動によりメモリーキャッシュを強制開放
(5) 1 分間休止
(6) Word Count を 5 回繰り返し実行
(7) 1 分間休止
(8) 入力データ，中間データ，出力データを削除
(9) 1 分間休止

3.4 評価結果

Regular サイズでの Hadoop，SSS の実行結果をそれぞれ**図 4**，**図 5**，**図 6**，**図 7**，**図 8**，**図 9** に示す．横軸は経過時間，縦軸は実験用クラスタ全体の消費電力である．点線で示しているのは，それぞれ入力データ投入開始，投入完了，Word Count の（開始，終了）× 5，データの削除開始，削除完了のタイミングである．SSS については，Reduce 処理の開始時点も示してある．

Hadoop でも SSS でも，ストレージの違いによる顕著な差異は見られない．

Hadoop と SSS との比較では，Hadoop が定常的に約 8kW の電力を消費するのに対して，SSS はベースラインとして約 5kW の電力を消費し，Map フェーズの間スパイク状に電力を消費する．これは，SSS の Map が中間データを定期的に分散 KVS の担当ノードに通信して格納しているためと考えられる．また，Reduce フェーズ開始とともに一段と高い値（約 8kW）を記録しており，主に Reduce フェーズで中間データを読み込む処理に電力を要すると推定される．

表 2 にデータ格納，Word Count 実行，データ削除に要した時間を示す．SSS は Hadoop に比べて約 10 倍データ格納が速いが，Word Count の実行では 40%程度遅く，削除では 2 倍遅い．データ格納が速いのは，SSS が用いている分散 KVS 実装である Tokyo Tyrant が持つメモリーキャッシュに格納され，その直後の Tokyo Tyrant の再起動時点で強制的に書き出されるためであると推定される．削除が遅いのは，SSS では中間データも分散 KVS 上に保持しており，削除データの件数が非常に多い（入力データ 8,721 レコード，5 回分の中間データ計約 21 億レコード，5 回分の出力データ計約 1 億レコード）ためであると推定される．一方で，Word Count の実行自体が遅いのは最適化

表 2 Regular - Time Comparison [sec]

	Store	WC avg.	Delete
Hadoop + ioDrive	179	228.0	14
Hadoop + SATA SSD	161	228.8	14
Hadoop + SAS HDD	156	228.0	13
SSS + ioDrive	16	319.8	27
SSS + SATA SSD	15	318.0	28
SSS + SAS HDD	16	319.0	29

表 3 Regular - Power Consumption [Wh]

	Store	WC avg.	Delete
Hadoop + ioDrive	192.7	482.5	15.1
Hadoop + SATA SSD	174.2	484.0	15.1
Hadoop + SAS HDD	168.7	482.6	14.0
SSS + ioDrive	19.0	450.3	36.2
SSS + SATA SSD	17.9	448.7	37.4
SSS + SAS HDD	19.1	449.6	38.8

が不十分であるためと推測され，Hadoop と SSS のベースラインの消費電力がそれぞれ約 8kW，約 5kW を示していたこともその傍証となる．正確性を期するには，各コアの稼働状況をモニタする必要がある．

また，**表 3** にデータ格納，Word Count 実行，データ削除に要した消費電力量を示す．消費電力量は台形公式を用いて近似計算したものである．概ね 7%，SSS の方が消費電力量が少ないことが分かる．

他方，Small サイズでの Hadoop，SSS の実行結果をそれぞれ**図 10**，**図 11**，**図 12**，**図 13**，**図 14**，**図 15** に示す．横軸は経過時間，縦軸は実験用クラスタ全体の消費電力である．点線で示しているのは，それぞれ入力データ投入開始，投入完了，Word Count の（開始，終了）× 5，データの削除開始，削除完了のタイミングである．区間ごとの実行時間，消費電力量も同様に，**表 4**，**表 5** に示す[*1]．

Regular サイズの場合と同様，Hadoop でも SSS でも，ストレージの違いによる顕著な差異は見られない．総データサイズが Regular の約 0.1% しかないにも関わらず，Hadoop は Regular と同程度の実行時間と消費電力量を記録している．また，定常的に約 8kW の電力を消費しているのも共通している．一方，SSS は，Hadoop より 2 桁速く，消費電力量も 2 桁少ない．これらの結果は，最初の Word Count 実行時に入力データセットを読み込んだ時点で，分散 KVS のメモリーキャッシュに対象データが格納されること，Map，Reduce タスクの起動オーバーヘッドが Hadoop に比べてかなり小さいことで説明できる．

4. 関連研究

大規模な分散データストアクラスタ上での MapReduce において，ストレージノードを電源停止させることにより

[*1] SSS の実行時間が非常に短いため，実行時間，消費電力，消費電力量ともに極めて精度が低いことに注意．

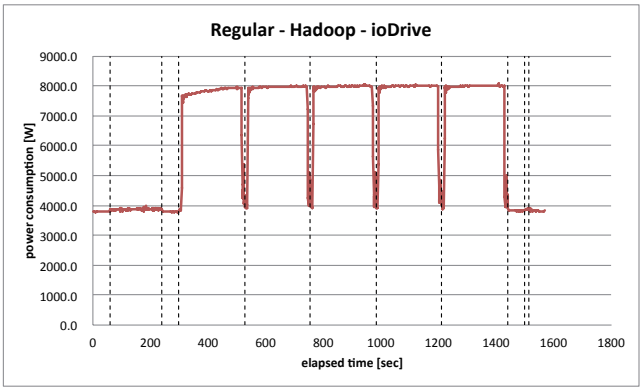

図 4 Regular - Hadoop - ioDrive

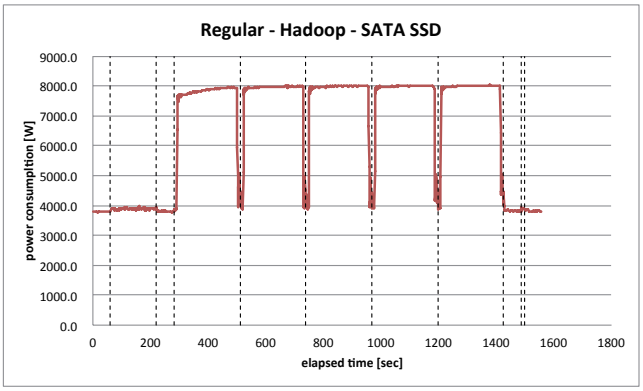

図 5 Regular - Hadoop - SATA SSD

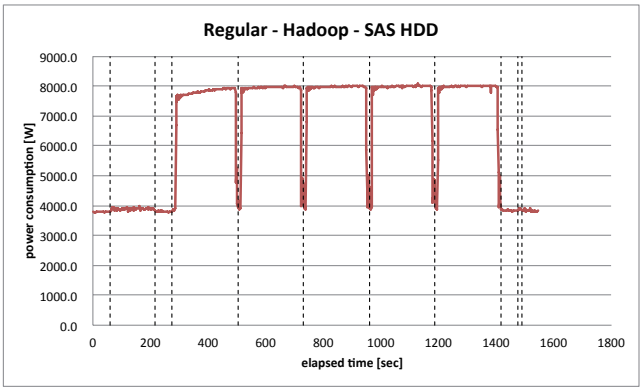

図 6 Regular - Hadoop - SAS HDD

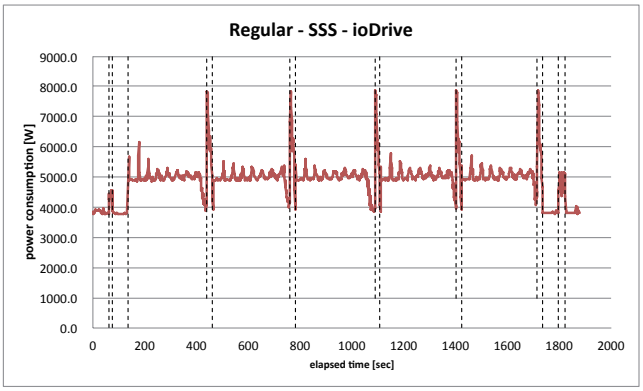

図 7 Regular - SSS - ioDrive

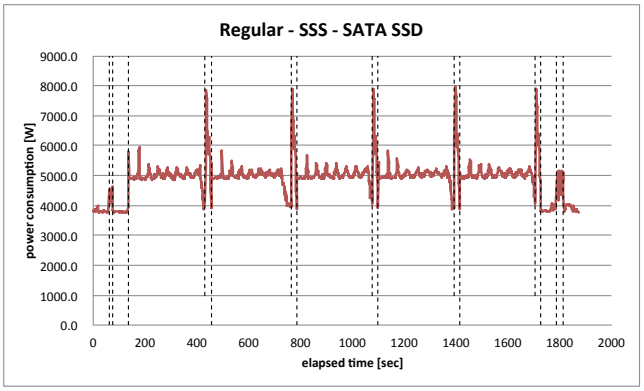

図 8 Regular - SSS - SATA SSD

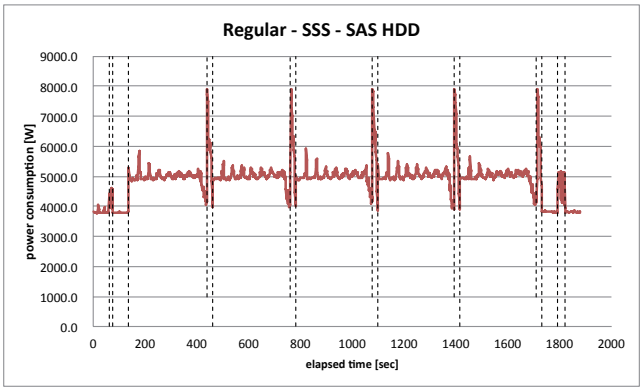

図 9 Regular - SSS - SAS HDD

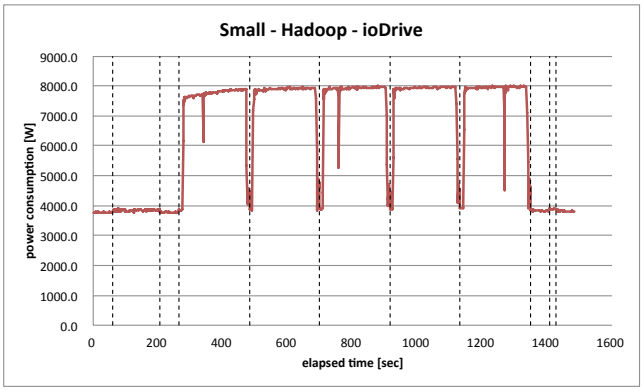

図 10 Small - Hadoop - ioDrive

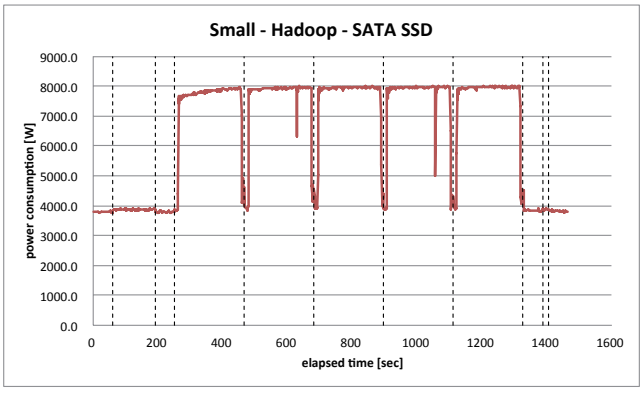

図 11 Small - Hadoop - SATA SSD

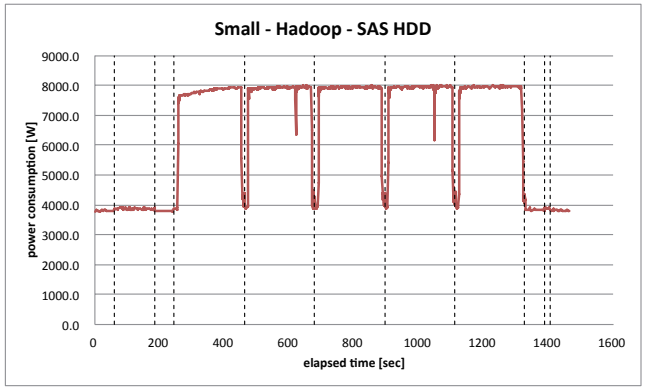

図 12 Small - Hadoop - SAS HDD

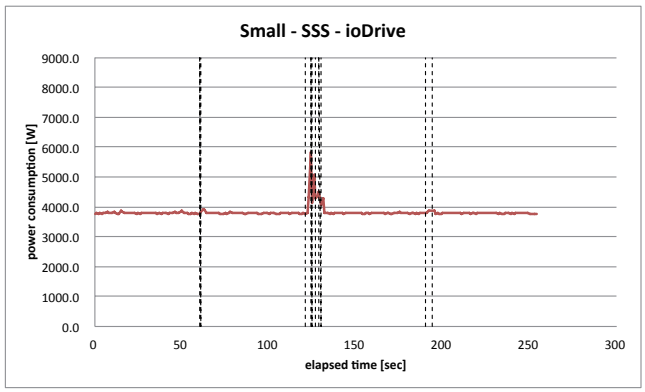

図 13 Small - SSS - ioDrive

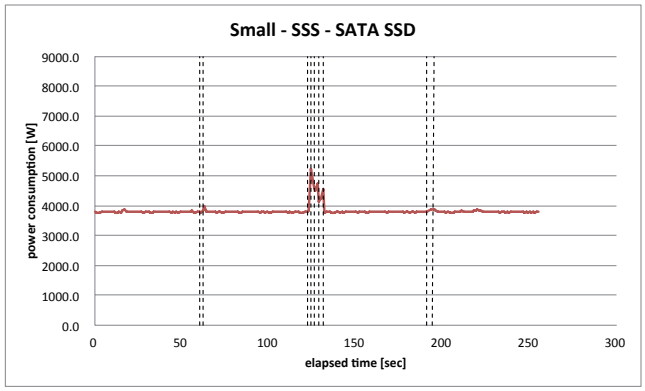

図 14 Small - SSS - SATA SSD

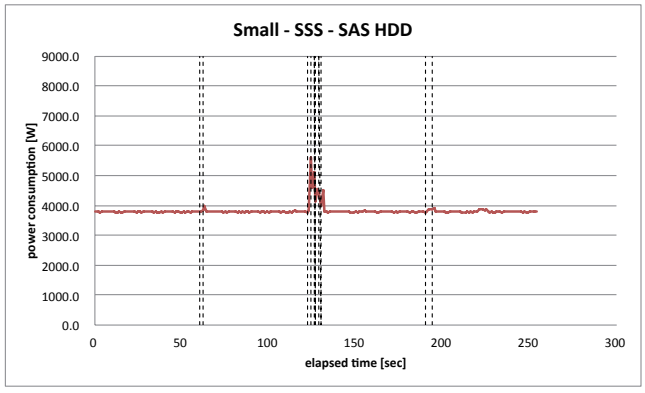

図 15 Small - SSS - SAS HDD

表 4 Small - Time Comparison [sec]

	Store	WC avg.	Delete
Hadoop + ioDrive	145	216.6	19
Hadoop + SATA SSD	131	215.0	20
Hadoop + SAS HDD	124	216.4	19
SSS + ioDrive	1	1.8	27
SSS + SATA SSD	2	1.8	28
SSS + SAS HDD	2	1.6	29

表 5 Small - Power Consumption [Wh]

	Store	WC avg.	Delete
Hadoop + ioDrive	154.8	452.7	20.5
Hadoop + SATA SSD	140.8	451.9	21.5
Hadoop + SAS HDD	133.5	453.7	20.5
SSS + ioDrive	1.1	2.2	4.3
SSS + SATA SSD	2.1	2.2	3.2
SSS + SAS HDD	2.1	2.0	4.3

消費電力量を削減する研究が数多くある．

Kaushik らの Green HDFS[12] では，hot zone, cold zone という 2 つの HDFS クラスタを用意し，頻繁にアクセスされるデータは常に電源が供給されている hot zone に配置される．書き込み容量を確保するために，Green HDFS では cold zone では同時に 1 ノードのみ電源を入れる．この方法では，cold zone に書き込まれた内容が後で頻繁にアクセスされる場合にアクセスが集中してしまう問題があると思われる．

Leverich らの Covering Subset Scheme[13] では，各データブロックのレプリカの一つを Covering Subset と呼ばれる電源停止させないノード群に格納する．この Subset はデータの可用性を確保するために電源を供給し，残りは電源を停止することで省電力化を図る．省電力化と引き換えに書き込み性能，容量等を犠牲にする．

Lang らの All-In Stategy[14] では，inactive な期間は全ノードの電源を停止にするポリシーを採る．処理すべきジョブがキューに貯まれば，全ノードの電源をオンにする．この場合，すべてのジョブが一旦待たされるため，インタラクティブなジョブの実行には適ささない．同様のケースについて，Chen らも調査 [15] している．

同じく，Chen らの BEEMER[16] では，インタラクティブジョブは少量のデータしか用いないことに注目し，常時電源供給された少数ノードからなるインタラクティブジョブ専用クラスタで実行する．それ以外の時間にセンシティブでないジョブは，All-In Strategy と同様にバッチ的に電源投入，ジョブ実行，電源停止をおこなうことで省電力を実現する．

小林ら [17] は，柔軟なデータ配置機構，複製へのアクセス転送，他ノードへのライトオフローディングなどの手法を分散 KVS 実装に導入することで，より多くのストレージノードを電源停止しつつ，性能を維持することを試みている．ただし，MapReduce などのアプリケーションでの有効性は明確ではない．

本稿の SSS では，ノードの電源停止をしない状態での消費電力を対象としている．hot なデータに繰り返しアクセスするようなワークロードでは，メモリーキャッシュが十分に利用できるため，高速かつ低消費電力で実行できる．

5. まとめ

われわれは実験用クラスタの消費電力を秒単位で計測・報告するシステムを構築した．その上で，Hadoop とわれわれが開発している MapReduce 処理系である SSS を用いて，English Wikipedia の全テキストデータを対象に Word Count を実行するベンチマーク実験を実施し，その結果を示した．SSS は，7.8GB 規模のデータセットに対しては，性能面でチューニングの余地がかなりあるものの，Hadoop に対して消費電力量で約 7%抑えられることが分かった．小規模・多数のデータセットに対しては，SSS はごく高速に動作することも改めて確認し，この場合，結果的に消費電力量がごく低く抑えられることも確認した．

電力特性が可視化されたことで今後のさらなるチューニングに役立てられるものと考えている．また，単純な Word Count のイタレーションではなく，K-means，PrefixSpan 法，疎行列ベクトル積などでのの電力特性の評価，複合的なワークロードでの評価なども今後の課題である．

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] Dean, J. and Ghemawat, S.: MapReduce: Simplified Data Processing on Large Clusters, *Communications of the ACM*, Vol. 51, No. 1, pp. 107–113 (2008).
[2] Apache Hadoop Project: Hadoop, `http://hadoop.apache.org/`.
[3] Hidemoto Nakada and Hirotaka Ogawa: SSS MapReduce, `http://sss.apgrid.org/`.
[4] 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 (1st International Workshop on Theory and Practice of MapReduce)*, pp. 754–761 (2010).
[5] 小川宏高，中田秀基，工藤知宏：合成ベンチマークによる MapReduce 処理系 SSS の性能評価，情報処理学会研究報告 2011-HPC-130 (2011).
[6] 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 論文集，pp. 21–29 (2011).
[7] 中田秀基，小川宏高，工藤知宏：MapReduce 処理系 SSS の PrefixSpan 法による性能評価，情報処理学会研究報告

2011-HPC-133 (2012).
[8] 高野了成，中田秀基，清水敏行，工藤知宏：クラウドを利用した電力可視化システムの構築，情報処理学会研究報告 2011-OS-119 (2011).
[9] 昆盛太郎，河西勇二，村川正宏，樋口哲也：消費電力の見える化とその評価・校正技術の開発，電気学会計測研究会講演予稿集 IM-11-036 (2011).
[10] 産総研プレス発表：電力可視化システムを低コストで構築，`http://www.aist.go.jp/aist_j/press_release/pr2011/pr20110902/pr20110902.html` (2011).
[11] Wikipedia Foundation, Inc.: Wikipedia:Database download - Wikipedia, the free encyclopedia, `http://en.wikipedia.org/wiki/Wikipedia:Database_download` (2012).
[12] Kaushik, R. T., Bhandarkar, M. and Nahrstedt, K.: Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System, *IEEE Second International Conference on Cloud Computing and Technology and Science (CloudCom)*, pp. 274–287 (2010).
[13] Leverich, J. and Kozyrakis, C.: On the energy (in)efficiency of Hadoop clusters, *SIGOPS Oper. Syst. Rev.*, Vol. 44, No. 1, pp. 61–65 (2010).
[14] Lang, W. and Patel, J. M.: Energy management for MapReduce clusters, *Proc. VLDB Endow.*, Vol. 3, No. 1-2, pp. 129–139 (2010).
[15] Yanpei Chen and Archana Sulochana Ganapathi and Armando Fox and Randy H. Katz and David A. Patterson: Statistical Workload for Energy Efficient MapReduce, Technical Report UCB/EECS-2010-6, EECS Department, University of California at Berkeley (2010).
[16] Chen, Y., Alspaugh, S., Borthakur, D. and Katz, R.: Energy efficiency for large-scale MapReduce workloads with significant interactive analysis, *Proceedings of the 7th ACM european conference on Computer Systems*, EuroSys '12, New York, NY, USA, ACM, pp. 43–56 (2012).
[17] 小林大，管真樹，大野善之，鳥居隆史：構成ノード電源停止によるシステム省電力化のためのインメモリ分散データストア設計，*Proceedings of the 3rd Forum on Data Engineering and Information Management* (2011).