スケーラビリティと高効率性を備えたクラウド基盤を実現するデータセントリック分散制御

菅　真樹† 小林　大† 鳥居　隆史† 小川　周吾† 板橋　康雄†
宮田美知太郎† 山川　聡† 長谷部賀洋†

† NEC システムプラットフォーム研究所
神奈川県川崎市中原区下沼部 1753
E-mail: †{kan@bq,daik@ay,t-torii@ce,s-ogawa@ak,y-itabashi@ cw,
m-miyata@if,s-yamakawa@if,y-hasebe@ah}.jp.nec.com

あらまし クラウドコンピューティングの流行により，多種・多数のサービスがデータセンター内のコンピュータシステム上で動作するようになりつつある．クラウドサービスにおいて低コスト化は重要な価値の 1 つであり，電力やハードウェア利用率などの観点による効率性向上が必要である．本稿では，従来の分散システムに含まれる多様な非効率性を解消するデータセントリック分散制御を提案する．本制御を実現するためにシステムに必要な 4 特徴，即ち，データとアプリケーションの PF 層における特性理解，多様なデータ構造，柔軟なデータ配置，柔軟な処理配置について延べ，これらを満たす分散システムアーキテクチャを提案する．大規模データ処理のユースケースを想定した模擬環境に対し，本稿で提案する制御を適用した最適化により，システム効率性を向上させる例を示す．
キーワード クラウドコンピューティング，分散システム，ストレージ，省電力化，効率化

Data Centric Distribution Control for Scalable and Efficient Cloud Platforms

Masaki KAN†, Dai KOBAYASHI†, Takashi TORII†, Shugo OGAWA†, Yasuo ITABASHI†,
Michitaro MIYATA†, Satoshi YAMAKAWA†, and Yoshihiro HASEBE†

† System Platforms Research Laboratories, NEC Corporation
1753, Shimonumabe, Nakahara-Ku, Kawasaki, Kanagawa
E-mail: †{kan@bq,daik@ay,t-torii@ce,s-ogawa@ak,y-itabashi@ cw,
m-miyata@if,s-yamakawa@if,y-hasebe@ah}.jp.nec.com

Abstract Many kinds of IT services are to be aggregated into a few large data centers in cloud computing era. The data centers demand drastic improvements for efficiencies such as power consumption and operation rates in addtion to current high scalability, because the efficiencies lead management cost reduction directly. In this paper, we discuss the efficient cloud platforms and provide an overview of a Data-Centric Distribution Control that we propose. Our suggestion includes that the efficient platform requires four properties: automatic recognition of characteristics of data and applications, diversity of stored data structure, and flexibility for distributed data and process placement structures. Our Data-Centric Distribution Control give the properties to cloud computing platforms. We also provide an experimental overview of influence of the four properties on the efficiencies.
Key words Cloud computing, Distributed system, Storage, Energy-efficiency

1. はじめに

仮想化されたコンピューティング資源をネットワーク越しに利用するクラウドコンピューティングの普及により，近い将来少数のデータセンター事業者によって，多数の個人や企業のシステムを運用する基盤を提供すると言われている．また，これらの個人や企業の扱うデータの総量は EByte オーダに達する見込みである [1]．従って，クラウドサービスの事業者は大規模

なデータに多種多様なアプリケーションを動作させることが出来るシステムを顧客に提供する必要がある．

現在，大規模なデータを処理するクラウド事業者が運用するシステムはスケーラブルであるが，非効率性も増加している，と考える．クラウド事業者は大規模なデータ処理に対し，MapReduce に代表されるような，データインテンシブスケーラブルコンピューティング (DISC) システムを構築している．安価な大量の計算機を利用してデータ処理を行う DISC により，安価でスケーラブルなシステムを構築することができる．しかし，現在の DISC ではその他の効率性が犠牲になっていることが知られている [2]. 例えば，我々の着目するハードウェア稼働率はシステムに備わる計算資源の利用効率である．計算資源の一部が SSD などの高速 I/O 装置，GPGPU や FPGA などのある傾向をもつ処理に優れたハードウェアを備えるシステムが登場している．このようなヘテロジーニアス性が増したシステムにおいてもこれらの HW の恩恵を受けられるためには，より処理の傾向や，各ノードの効率を意識してシステム資源を利用する必要がある．また，電力効率性の悪さは，電力購入コストや電源・冷却設備維持などのサービス運営コストの上昇を招くため，電力視点による効率性向上も必要とされる．また，これらの効率化を，大規模データ処理だけでなく，多種多様なアプリケーションに対して，システムとして自律的に適応する必要がある．

我々は，システムの非効率性のうち，リソース稼動非効率性，データアクセス非効率性，ヘテロ非効率性の 3 点に注目する．これらの非効率性は，データレイヤとアプリケーションレイヤの乖離に依存するものと考える．本稿で提案するデータセントリック分散システムアーキテクチャでは，従来，この乖離によって困難であった，データとアプリケーションの特性理解，制御可能な多様なデータ構造，制御可能な柔軟なデータ配置，システム構成要素の動作状態を反映した処理配置，の 4 特徴を備えることでこれらの非効率性を解決することを目指す．本アーキテクチャは，これら 4 特徴を満たすため，各ノード上で動作する機能コンポーネントによるシステム情報解析・収集，ストレージ層実装のプラガブル構成，多段階に拡張したコンシステントハッシング方式，運用管理エージェントによる自律制御機能を保持する．

また，提案する研究アプローチの有用性を確認するために，データ処理における 3 つのユースケースの模擬環境を構築した．そして，模擬環境上で 4 特徴を用いた制御を行うことにより，得られることが期待される，システム効率性向上効果について示す．

本稿は以下の通り構成される．2. では，DISC システムにおける課題についてユースケースを織り交ぜ説明し，効率的な DISC システムに必要な特徴を明らかにする．3. で我々の提案するデータセントリック分散システムの基本的な設計指針を示し，4. ではユースケースを想定した模擬環境にデータセントリック分散制御を適用した場合の効率性向上効果について述べる．5. で関連研究について述べ，6. でまとめる．

2. DISC クラウドシステムにおける課題

現在のクラウドプラットフォームとなる分散システムは，スケーラブルであるが非効率である．増大する多くの顧客システムを安価でスケーラブルに格納する設計を強調したため，DISC システムは，稼働率や消費電力等，その他の効率性について課題のあるシステムが構築されている．このような非効率性は，システム維持コストや消費電力に還元される. クラウド時代のプラットフォーム運営者は，サービス運営コストを可能な限り低減し，コスト競争力のあるサービスを提供することが求められるため，分散システムには抜本的な効率化が必要となっている．

2.1 ユースケース

典型的な DISC システムのユースケースを元に，現在の DISC システムの非効率性について述べる．

Apache Hadoop [3] に代表される典型的な DISC システムは，数千，数万台の安価なサーバ（ノード）により構成され，ユーザから格納されたデータセットに対して任意の処理を行う．データセットは，単純なデータ配置アルゴリズムに従い全ノードに対して冗長性を考慮しながら分散格納される．データセットに対する一連の処理は，各データごとの処理に分解される．分解された処理は，当該データの格納されたノードにおいて処理される．処理結果はノード間で統合され，次の処理あるいは処理呼び出し元に返る．処理対象データは，プロパティの集合をひとつのデータオブジェクトとして処理される．データ配置アルゴリズムは，データ単位に一意のキープロパティに基づき，コンシステントハッシング [4] や水平分割 [5] が用いられる．各ノードでは，データ格納性能を重視しシーケンシャライズされたデータ単位で格納される．一連の処理は，MapReduce 等のフレームワークで記述されることで，容易に分解される．

単一データごとの単純な処理の集合の羅列によりデータに処理を与える DISC システムでは，高いシステムスケーラビリティが得られる一方で，その他の効率を犠牲にしていることが知られている．非効率性の一部は，スケーラビリティのための単純さによる．データ格納性能を重視したデータ構造では，アプリケーションによってはランダムアクセスによる性能劣化が著しい．データ構造もこの分析アプリケーションに向けた構造であれば，データ I/O 量を削減できるため，アプリケーションが必要とする最小限のアクセスと比較すると非効率な I/O が発生することになる．同様に，システムの一部のノードが GPGPU や FPGA 等の処理高速化ハードウェア (HW) が利用可能であっても，従来のシステムでは，当該 HW の配置を反映していないデータ配置に処理の配置が依存し，処理高速化 HW の活発な利用が難しい．また，必要あればデータ構造をその都度専用 HW 用に転送・変換する必要があり，削減されたデータ転送オーバヘッドが再発する．

あるいは，非効率性は，処理特性の無理解によっても生み出される．クラウドプラットフォームに対する利用形態によっては，ある時間帯においてはわずかなアプリケーションだけが動作するようなことが考えられる．この際に，データ格納容量，データ処理性能として必要される最小数以上のノードが稼働し

続けていることは非効率である．また，収集格納しているデータが多くの同一データから構成されたり，多くのデータがアクセスされない場合に，RAW データのまま保持しておくことは必要以上のストレージ容量を利用することになるため，必要な HDD 数の増加や分散ストレージノードの増加に繋がり，非効率である．また，データ間の相関を必要とする高度な分析アプリケーションでは，分散ストレージを構成するノード間で多くのノードと通信をしてデータを取得しなければならないが，単純なデータ配置アルゴリズムでは，アプリケーションごとのデータ間操作の特性が反映されず，非効率なデータ転送が頻発する．

2.2 システム非効率性

このように，従来のクラウドプラットフォーム上での典型的な大規模データ分析システムは，様々な非効率性を持つ．本稿では DISC システムの非効率利用のうち次の 3 つの非効率性に着目する．

リソース稼働非効率性 とは，システムとして必要のないデバイスが動作することを指す．デバイスの不必要な動作は消費電力や発熱量を増加させる．

データアクセス非効率性 とは，ある処理が必要とするデータアクセスに対し，最低限以上のネットワーク I/O やストレージ I/O が発生することを指す．例えば，処理対象データは処理計算を行うプロセッサ近くに予め配置されていれば，冗長なネットワーク I/O は発生しない．また，ネットワークや記憶素子のアクセス特徴に合致しないアクセスによっても非効率性は生まれる．例えばパケット長と不一致のデータ転送や HDD へのランダムアクセス等である．これらのデータアクセス非効率性は，I/O による消費電力の増加，スループットやレイテンシ性能の低下を引き起こす．

ヘテロ非効率性 とは，分散システムを構成するノードが，不均質な特性を持つことによって引き起こされる非効率性を指す．近年，汎用 GPU や FPGA，SSD などの，ある傾向をもつ処理に対しては低電力かつ高速に動作するハードウェアを用いた処理が注目されている．しかし特殊なハードウェアは高価であり，また用途が限定されるため一部のノードにのみ備わることが想定される．このようなハードウェアに合致した処理は当該ノードで処理されるべきであり，通常の処理は安価なノードで処理されるべきであるが，現在の DISC システムではヘテロ性を有効活用できていない．

これらの 3 つの非効率性は DISC システムの持つスケーラビリティ主導の設計により引き起こされている．

リソース稼働非効率性を引き起こす要因としては，必要最低限以上のデータ容量確保や, 非効率なノード停止処理などが当てはまる．従来の一般的なクラウド向けシステムでは，データ保持形式やノードの動作状態の制御をスケーラビリティを中心に行っており，動作するアプリケーションやデータの特性を認識していなかったため，このような非効率性を解消できていない．

データアクセス非効率性を引き起こす主な要因としては，データの不適切な配置やデータ構造ミスマッチがある．従来の一般的なクラウド向けシステムでは，アプリケーションを動作させるノードをシステム側で自由に選択できず，またアプリケーションが利用するデータの構造や配置場所を自由に制御することが困難であったため，このような非効率性を解消できていない．

ヘテロジニアスリソース利用非効率性を引き起こす主な要因は，リソース稼働非効率性と同様にデータの不適切な配置やデータ構造ミスマッチである．特殊ハードウェアの利用効果を最大限に活かすには，このようなハードウェアを利用するアプリケーションが処理対象とするデータを，特殊ハードウェアを保持するノードに適切な形で保持させる必要がある．しかし，従来の一般的なクラウドプラットフォームでは，システム側でデータがどのようなアプリケーションで用いられるかを認識しておらず，またデータ構造を自律的に制御出来ないために特殊ハードウェアの効果を最大限に活かすことが出来ていない．

2.3 効率的な DISC システムに必要な特徴

DISC システムがよりデータセンター事業者にとって価値あるシステムとなるために，従来のスケーラビリティ性を維持しつつ，これら 3 つの非効率性要因を排除することが必要となる．我々は，このような非効率要因が，データセンター事業者であるプラットフォーム（PF) 層とその顧客であるアプリケーションプロバイダー層の乖離によるものと考え，PF 層にさらに次の 4 つの特徴を新たに備えることでこの問題を解決する：

（1） PF 層でのデータとアプリケーションの特性理解
（2） PF 層で制御可能な多様なデータ構造
（3） PF 層で制御可能な柔軟なデータ配置
（4） システム構成要素の動作状態を反映した処理配置

このような 4 つの特徴を備えることで，従来より非効率性を排除したデータ配置・処理配置が実現できる．その上で，システムを構成する各ハードウェア要素・ミドルウェア要素ごとに，既存手法による効率化を行うことでシステム全体の非効率性を大きく低減できる．既存手法としては例えば，電力消費量の観点に基づく非効率性を解消するために，分散システムの構成ノードの電源を停止したり，ノードの構成要素（CPU や HDD など）の動作モードの変更や，構成要素の停止の手法が知られている [6].

4 特徴を実現したシステムを用いて，これらの既存手法の効果さらに高める制御を実現できる．例えば，データの配置場所を柔軟に変更することで，ノード停止時間を増大させる制御アプローチ [7] や，ストレージより上位のレイヤの特性情報を用いて電力効率を上昇させる [8] といった方法が 4 特徴の一部を用いて行われている．

3. スケーラビリティと高効率性を備えたデータセントリック分散システム

データセントリック分散システムは，データの内容や利用傾向を中心にシステムを制御することで 2.2 で述べた非効率性を解消する DISC プラットフォームである．スケーラビリティと高効率性を備えたクラウドプラットフォームを提供するため，2.3 で述べた 4 特徴を満たす分散システムアーキテクチャおよび，高効率化を実現する制御手法の双方を提供する必要がある．

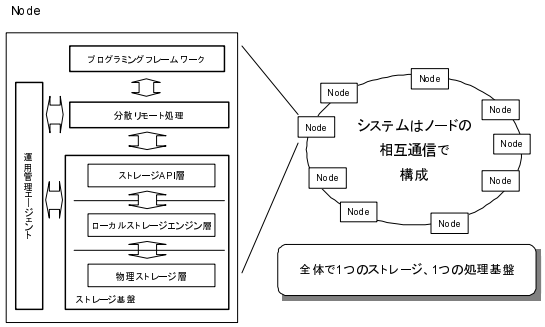

図 1　データセントリック分散システムの全体構成と機能コンポーネント

Fig. 1　Architecture of distributed system

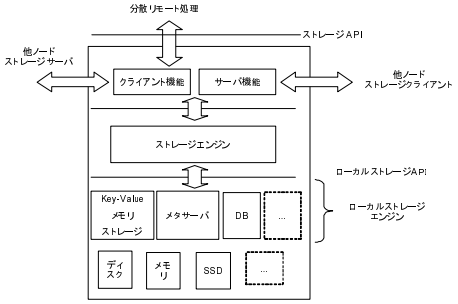

図 2　3 層からなるストレージ基盤の構成図.

Fig. 2　A Storage Platform consists of three layers.

本稿では，将来の制御手法の適用を踏まえつつ，前者の分散システムアーキテクチャを中心に議論する．

3.1　全体構成

データセントリック分散システムは各ノードで動作する機能コンポーネントが協調動作することにより特徴（1）から（4）を満たすよう実現される．ノード数は数百から数万を想定し，各ノードは様々な仕様を持つ計算機で構成される．これらはデータセンター内に配置され，標準的なネットワークにより接続されている．多くのノードはコモディティハードウェアで構成されているが，一部のノードは SSD や GPGPU など特殊なハードウェアを備えていても良い．

データセントリック分散システムの構成を図 1 に示す．各ノード上の機能コンポーネントは，ストレージ基盤，分散リモート処理・プログラミングフレームワーク，運用管理エージェント，から構成される．ストレージ基盤は，ノード間の分散協調によりデータを永続化し格納・提供する機能を実現する．分散リモート処理とプログラミングフレームワークは，ユーザプログラムを複数の分散ノード上で動作させるための機構である．各ノード上で動作する運用管理エージェントは協調し，各機能コンポーネントからの情報を元に機能コンポーネントやノード構成要素（HDD など）に指示することによって，システム全体の管理制御を行う．

3.2　多様なデータ構造と柔軟なデータ配置の実現

スケーラビリティ，特徴（2）多様なデータ構造，及び特徴（3）柔軟なデータ配置は，ストレージ基盤により提供する．特徴（2）を満たすため，ストレージ基盤は，ローカルストレージエンジン層の実装をプラガブル構成を取る．特徴（3）とスケーラビリティ両立のため，データの配置場所制御は，コンシステントハッシング方式の拡張により制御する．

ストレージ基盤の構成を図 2 に示す．ストレージ基盤は段階的な抽象度からなる 3 層で構成され，多様なデータ構造へ対応可能とする．物理ストレージ層は，HDD やメモリ等物理的な媒体を制御する．ローカルストレージエンジン層は，DBMS エンジンや Key-Value ストアなどのソフトウェアロジックである．ストレージアクセス API 層により，分散環境制御と，データへの格納 API を提供する．ローカルストレージエンジン層では，複数の特徴を持つロジックを用意し，データ集合ごとにエンジンを選択するプラガブル構成をとる．これにより，データの特性情報を用いたデータの構造変更を可能にする．

データの特性情報は，データのサイズ，アクセスサイズ，データへの R/W 比率，データにアクセスするプログラム名，データ間の重複度合い等の情報である．ストレージエンジンは，データ毎にこれらの特性情報に適応した特徴を持つローカルストレージエンジンを選択し，データを格納する．これらのデータ特性情報を取得するための構成は 3.3 に後述する．

柔軟なデータ配置制御を実現するために，コンシステントハッシング方式におけるハッシュレンジに複数のノードから構成されるグループを割り当てる方式をとる．図 3 に本方式の構成例を示す．本方式のデータ配置場所の決定は 2 段階の処理で行う．まず，各データごとに一意の識別子をキーとしたコンシステントハッシュ方式により，該当データを管理するグループを決定する．そして次に，グループ内で，構成ノード消費電力や，ネットワークトポロジを考慮した詳細な動的データ配置制御・複製制御を行う．グループ内の配置制御には，分散索引方式，集中管理方式，コンシステントハッシング方式などを用いることが出来る．グループを構成するノード数を最大でも数百台に抑えることによって，グループ内でのシステム管理情報の同期ボトルネックを防ぐ．また，後述する各データへのメタデータは，各グループ内で閉じた二次索引を分散索引構造により構成することで，主識別子以外のデータ検索に対応する．

このような 2 段構成により，上段のコンシステントハッシュでスケーラビリティを保ち，かつ 2 段目以降のグループ内配置制御により柔軟な配置を提供する．なおこのような多段構成の弊害となる，アクセス時の多段ホップは，索引構造の非同期複製（経路キャッシュ）により軽減することを検討している．

3.3　データ，処理，構成要素の特徴理解

特徴（1）データとアプリケーションの特徴理解，及び特徴（4）要素状態を考慮した処理配置の実現は，機能コンポーネントによるシステム情報解析・収集と，運用管理エージェントによる自律制御により実現する．

データとアプリケーションの特徴理解は，メタデータ情報及びアクセスログ解析により行う．ストレージ基盤では，データを複数のメタデータ（プロパティ）のついた**オブジェクト**として扱う．各オブジェクトに対し，ユーザは API またはプログラミングフレームワークを介して各種メタ情報を付与することが

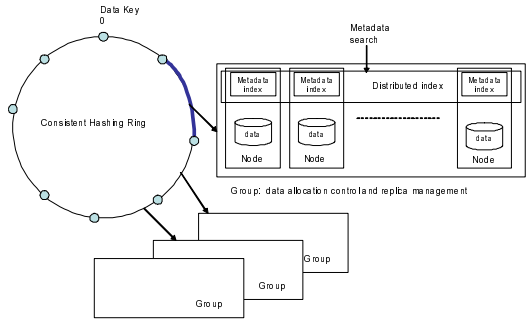

図 3　柔軟なデータ配置のためにハッシュレンジにグループを割り当てるデータ配置管理

Fig. 3　Multi-tier Distributed Hashing for Flexible Data Distribution

できる．各ノードでは分散リモート処理機構において，データに対する処理実行履歴をアクセスログとして記録し，アクセスログ解析によりデータに対する特徴情報に変換する．

また，ストレージ基盤コンポーネントは，前述した二次索引によるメタデータ情報を基にしたオブジェクト探索機能とともに，類似メタデータを持つオブジェクト群を簡易的なクラスタリングアルゴリズムにより発見することで，データに対する処理の特徴情報推定を行う．運用管理エージェントは，これらの特徴情報に基づいて，データ配置制御やローカルストレージ割り当ての制御，分散リモート処理機構におけるスケジューリング機構の制御を行うことにより，データとアプリケーションの特徴に基づいた分散システム制御を行う．

また，特徴（4）を満たすため，分散リモート処理においてスケジューリング機能を備え，各プログラムに対して最適なノードおよび実行方式を選択して，プログラムを実行する仕組みを備える．最適なノードは，前述したデータの特徴情報の他に，プログラム内でアクセスするデータの転送コスト，各ノードの実行時の負荷状況等の要素で決定される．また，プログラム実行方式の制御として，例えば，多数のデータへ同時に処理を行う際，重複したデータが複数存在した場合に，1つの処理結果を重複データに使いまわすような実行方式最適化を実現可能とする．

4.　配置柔軟性と特性理解の効果

本節では，本稿で提案する研究コンセプトの有用性について示す．大量データ処理における 3 つのユースケースの模擬環境を構築し，データと処理の配置柔軟性に基づく制御を適用することにより得られる，システム効率性の向上可能性について示す．

4.1 では，分散データ処理において，データの配置状態を変更した場合における，システム動作特性の差について，ネットワーク転送量に着目して示す．4.2 では，重複排除機能を備えたストレージ上でのデータ処理を行い，ストレージレイヤにおけるデータ格納特性を考慮したデータの配置制御や処理配置制御により，ディスク利用量や処理実行時間に差があることを示す．4.3 では，ヘテロジニアスリソース非効率性として挙げた，GPU と CPU が混在したクラスタ環境におけるデータ処理を行い，GPU 処理性能のポテンシャルを発揮するために必要なデータ配置制御について示す．

4.1　データの分散配置状態の違いによるシステムの動作特性差

本節では，データの分散配置状態が異なるときに，システム全体の動作特性が異なる例を示す．

評価ベンチマークとして，8 ノード並列で動作する並列 Bitonic ソートプログラムを用いる．そこで，本アーキテクチャを用いたデータ配置制御により，データ配置を最適化できた場合と，非効率に配置された場合両者について，ベンチマーク実行時のシステム動作特性を比較する．

データ配置最適化は，並列 Bitonic ソートにおいては最初のローカルソート実行のデータが，各計算ノードのローカルストレージに格納できている状態とする．**非効率データ配置**は，計算対象のデータが，計算ノードとは別の 1 台のストレージサーバに格納されている状態とする．前者は計算ノード各々のローカルファイルシステム上にデータを事前に格納しておき，後者は NFS サーバを別途用意してデータを格納することで模擬環境を構築した．これらの両者において，それぞれベンチマークを実行し，実行中の各ノードで IOSTAT,VMSTAT などにより負荷状況を監視した．

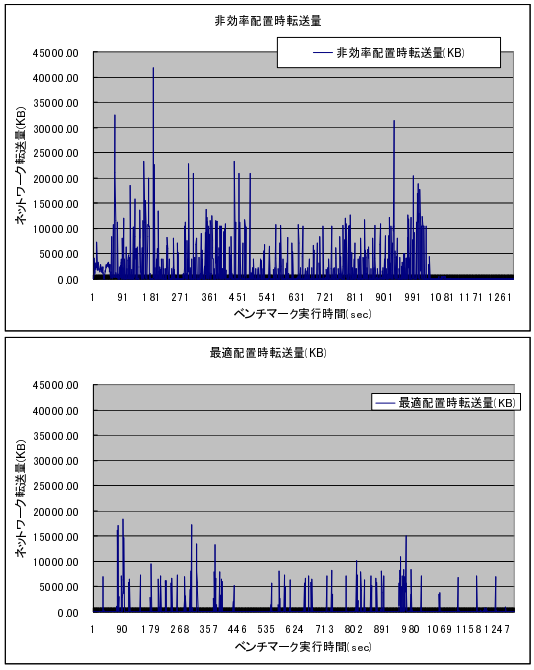

図 4　データ初期配置状態の違いによる時系列ノード間通信量

Fig. 4　Time-series network traffic amount with data allocation types

	各ノードデータ転送総量(MB)
非効率データ配置	2333.18
データ配置最適化	659.42

表 1　データ初期配置状態の違いによるネットワーク転送量差

Table 1　total of network traffic with data allocation types

データ最適配置状態と，非効率配置状態のそれぞれにおいて，ベンチマーク実行中のノード間ネットワーク転送総量を表 1 に，ベンチマーク実行時の各時間におけるネットワーク転送量小計について図 4 にまとめる．表 1 の通り，ネットワーク転送総量として約 1.6GB の差があった．これにより，もし各ノードやスイッチにおいて転送パケット量に応じた電力消費差があるならば，電力消費量削減が期待できる．

また，本実験における非効率データ配置状態では，データ I/O が NFS サーバに集中したにも関わらず，本試行では実行時間は非効率データ配置時の方が短かった．この理由は，本評価でので並列ソートのレコード数は 8,000,000 レコードにすぎず，1 レコードあたり 4KB のデータとしてもデータ容量として 32GB にしか達しないため，NFS サーバの I/O ボトルネックなどが発生しないためである．本稿の想定する大規模 DISC システムではより大量のデータ I/O が発生するため，ストレージノードにおけるデータ I/O ネックが発生して，データ配置によって性能差が発生してしまうことが想定される（その例については 4.3 にて後述する）．

また，ネットワークスイッチの電力消費量はその提供するバンド幅によって消費電力量が異なる．従って，本評価におけるデータ配置の制御による，ネットワーク転送量の削減により，ノード間を接続するスイッチを低いバンド幅のものを用いることなどにより消費電力量を削減出来ることが期待される．

4.2 重複排除可能なストレージ上での大量データ処理

本節では，3.3 で述べた，データ，処理，構成要素の特徴理解に基づく制御によって得られる効果について示す．

本評価では，ユースケースとして，大量データ処理の代表的な事例の 1 つである，ログ解析処理を想定する．また，データ容量効率向上のため，分散ノードのローカルストレージ層において重複排除機能が提供されるものとする．このローカルに閉じた重複排除機能という構成要素特性に基づいて，データの特徴傾向としてデータの内容に基づいた配置制御を行った場合の，データ容量効率向上および，ログ解析処理の性能を向上について示す．また，特徴（4）の要素状態を考慮した処理配置の実現により，重複データに対する処理結果の使い回しによる処理高速化を適用した場合の効果についても示す．

評価システムは，Linux サーバ 8 ノードから構成され，ログ解析処理としては分散 Grep 処理 (GNU grep -c 相当) を行う．分散 Grep 処理プログラムは，処理実行中のネットワークボトルネックを最小化するために，それぞれのノードが保持するファイルの Grep 処理を担当する事とする．また評価システムにおける重複排除機能は，各ノード内でファイルシステムのハードリンクにより模擬し実現する．本評価で想定する重複排除ストレージは，各ノードのローカルで実現され，ノード間にまたがる重複データについては，冗長して格納される．Grep 処理対象データは，1 ファイルあたり約 10MB のテキストファイルを 50000 ファイルとする．これらには重複して同一のデータが存在することを想定し，50000 ファイルは 10000 種類ファイルから構成する．また，実際はデータの種類によってファイルの重複する確率が異なるため，10000 種類のファイルのうち 2000 種類のファイルは 20 個重複し，他の 2000 種類のファイルは 2 個重複し，その他は重複が存在しないこととする．アプリケーションレイヤからはこれらのファイルは全て別のファイルとして扱う．

このような環境において，一般的な方法として各ノードにラウンドロビンで分散配置する方法 (**Round Robin**) と，データの内容に基づいて配置ノードを決定する方法 (**Content base**) の 2 種類で，Grep 処理時間とデータ格納容量の比較をおこなった．また，Grep 処理方法は，重複排除機能を使わない場合（**Standard Grep**) と，重複排除機能を用いるが処理レイヤでの最適化を行わない方法（**NAO**），同じく処理レイヤの最適化を行う方法（**AO**）の 3 つの方法を比較する．NAO 方式は，Grep 処理を全データファイルに対して行うが，AO 方式はノード内で同一内容のデータファイルに対する Grep 処理は行わず，内容ごと 1 つの Grep 結果を複製する方式で，本アーキテクチャにおける処理レイヤの内容を PF レイヤで把握した最適化制御を行う場合を想定したものである．

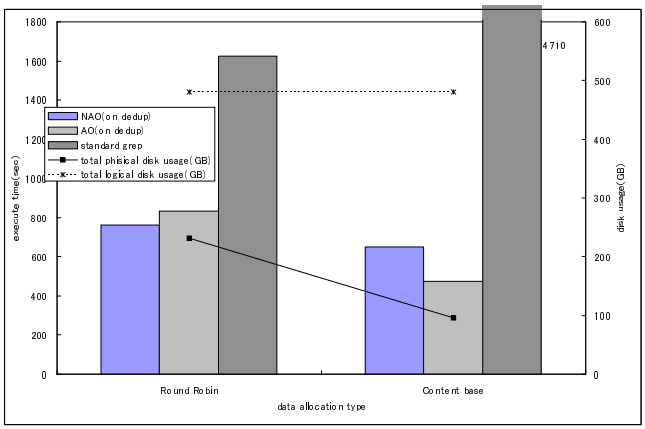

図 5 データ配置制御適用時の分散 Grep 処理実行時間およびディスク利用容量

Fig. 5 execution times of distributed grep and disk usages with data allocation controls

図 5 に Grep 処理実行時間およびディスク利用容量を示す．データ配置方式によらず，ストレージレイヤにおける重複排除により物理容量削減効果が得られる．ただし，その容量削減効果は，Round Robin 方式が約 52% に対し，Content base 方式により約 80%となる．また，分散 Grep 処理の実行時間については，重複排除ストレージを用いることで約 54%の実行時間短縮が見られ，ContentBase 方式によるデータ配置制御により，更に 15%の実行時間短縮が得られる．また，AO 方式による処理レイヤ最適化を適用する方法が最も優れたパフォーマンスを示し，通常のストレージに対してラウンドロビンで分散配置する方法と比較して，約 71%実行時間を短縮することが出来た．

これらの結果により，データ配置の自由化および，データの特性に応じた分散配置，処理内容を PF レイヤで把握した処理制御により効率化が可能であることのポテンシャルを示されたと言えよう．本節の結果は，ContentBase のデータ分散配置により，ノード毎の I/O 量がほぼ均等になるよう制御されたこ

とに基づくものであると考えられる．今回の Grep 処理のような I/O 負荷に偏ったアプリケーションであれば，I/O 負荷均等の配置方式が最も優れた結果を示す．しかし，アプリケーションの特性によってデータの適切な配置手法および処理の分散配置方法は異なり，2. で述べたように PF 層でのデータとアプリケーションとの特性理解とそれに基づいた制御が必要であると言えよう．

4.3 ヘテロノードクラスタにおける負荷分散

また，クラスタを構成するノードの性能や特性が異なる場合に，本稿で提唱するデータと処理配置の両方の制御を行った場合の有効性について評価を行う．GPU と CPU が混在するヘテロジニアスクラスタにおける処理分散制御として，アプリケーションの特性を理解し，GPU で実行可能（かつ高速）な処理を GPU ノードに多く割り当てる制御が考えられる．本評価では，このような処理配置制御だけでなく，処理配置制御をも踏まえたデータ配置制御の必要性について示す．

本評価システムは，CPU ノードが 8 ノード，GPU ノードが 1 ノード存在するヘテロジニアスクラスタ環境とする．ベンチマークアプリケーションは，大量データに対するソート処理とする．処理対象のデータ量は，576(64*9) ファイルとし，1 ファイルあたり 64M 個 (容量 256MB) のデータを保持する．実行する処理は，データファイルを各々READ し，ソートした結果をファイルとして出力する．

この場合に，GPU ノードに割り当てる処理量を変化させて実行時間を比較する．この際に，データの配置を各ノードに均等に分散させた場合と，GPU への処理割当量に応じて GPU ノードに集中して配置するよう制御した場合について比較する．

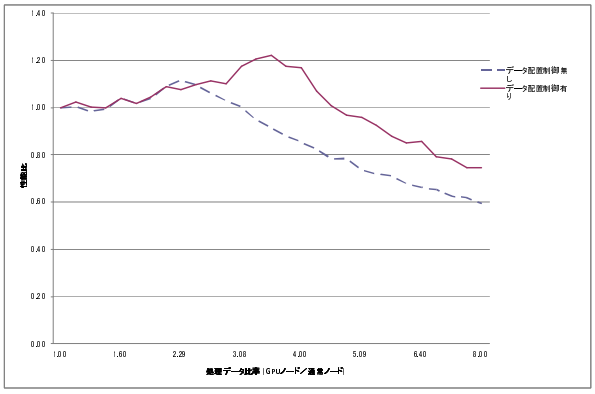

図 6　GPU ノードへの割り当てに応じたデータ配置制御の有無に対する性能差

Fig. 6　Performance of data allocation control corresponding to allotate jobs to GPU node

結果を図 6 に示す．本稿で用いたソート処理プログラムにおいて，GPU ノードの性能は CPU ノード性能の 3.5 倍程度である．従って，最も適切に制御された場合には，GPU:CPU が 3.5:1 の割合で処理を担当し，担当ノード上に処理対象データが格納されていれば期待される約 3.5 倍の性能を得ることが出来る．しかし，データの配置制御が適切に行われていない場合には，GPU 処理性能に応じた割り当てを行っても期待される性能が得られず，CPU ノードと GPU ノードの割当量を均等にした場合よりも性能が劣化さえしていることがわかる．この理由は，GPU ノードが他の CPU ノードからネットワークを介してデータの I/O を行わなければならないことと，またそのネットワーク越し I/O が CPU ノードにおける計算用の I/O と競合するためである．

このような結果により，ヘテロジニアスリソースクラスタにおいては，処理の配置だけでなく，データの配置についても適切に行うことが出来なければ，期待性能を得ることができず，本稿で提案するデータ配置と処理配置双方の自由化の必要性が示されたと考える．

本稿で提案するアーキテクチャにおいて，このようなデータ配置制御を実現するためには，データに付与されたメタデータからデータの利用特性をアクセスログ解析から導出しておく．そして，そのデータはある処理 A を実行される傾向にあり，処理 A は GPU 上で実行するためのエンジンを持ち，過去の性能結果で CPU ノードと比較して約 3.5 倍の性能を持つことを認識していれば，本節実験のように 3.5:1 の割合でデータを事前に割り当てておくような制御が可能となる．

5. 関 連 研 究

クラウド以前のシステムでは，データ格納を行うストレージシステムに対し，ストレージ装置にインテリジェント性を加え，格納されたデータの持つアプリケーション層の特性を応用した動的な管理・制御を行う研究が行われてきた [9], [10]．また，ストレージシステムの省電力化研究についても幾つか行われている [7], [8], [11]．多くの研究は，HDD 利用の時間的・空間的局所性が低いことを前提に，これらの局所性を一時的に高めた最適化制御を行い HDD やノードを停止させる，又は停止時間を延長させることによって省電力化を実現している．我々の研究ではデータアクセスに加え，付随するデータの処理も同様に考え，アプリケーションレイヤ情報を利用し局所性を向上する制御と見ることができる．

一方でコモディティなハードウェアによる DISC システムの実現 [3], [12], [13] により，データ格納資源と計算資源の結合が再び行われているが，分散制御が主眼となっている．例えば Google MapReduce では，Sawzall と呼ばれる言語で処理を記述することにより，ユーザが持つ処理の分散性情報を明示することで，分散制御を単純化している．しかし，本稿で述べた多様な効率化のための情報は，更なる機構によりアプリケーション層から引き出す必要がある．

また，ストレージレイヤにおいてアプリケーションレイヤの情報を取得し，ストレージ層での最適化制御を行う試みが行われている [14]〜[19]．これらの試みの多くは性能向上を目指したものであるが，省電力化制御に適用する試みも行われている [20], [21]. これらは，ストレージレイヤでの制御に注力しており，システム全体の制御という観点での研究が必要である．

処理の分散制御手法として，ヘテロ環境のクラスタにおける処理に関する研究 [22], [23] が行われている．これらは各ノードの性能差の一軸でヘテロ性を表現しており，処理の特殊性や消

費電力など多様な観点の研究がさらに望まれる．また, [24], [25] において,GPU で構成されたヘテロ環境クラスタおよび GPU の消費電力に着目した研究が行われている．

6. おわりに

本稿では，データ中心の制御によりシステムの非効率性を解消するデータセントリック・コンピューティング・分散システムアーキテクチャについて述べた．非効率性を解消するためにシステムに必要な 4 特徴，つまり，データとアプリケーションの PF 層での特性理解，多様なデータ構造，柔軟なデータ配置，柔軟な処理配置について延べ，これらを実現するためのアーキテクチャコンセプトを提案した．

本稿で述べた研究コンセプトの有用性を示すために，データ処理における 3 つのユースケースの模擬環境を構築した．これらの模擬環境に対し，本稿で提案する柔軟なデータ配置制御を適用することによる効率性向上ポテンシャルを示した．例えば，分散 BitonicSort 処理の実行時にデータの配置制御の適用により，システム全体のネットワーク通信量が約 72%削減されることを示した．また，重複排除ストレージ上での分散 Grep 処理では，データの内容に応じた分散配置により I/O 効率が改善され，かつアプリケーションレイヤでの最適化を行うことで約 71%の性能向上効果が得られることがわかった．さらに，GPU ノードを含むヘテロクラスタ環境における負荷分散制御では，GPU ノードにおける処理性能に応じた負荷割り当てと共にデータの最適配置を行うことで GPU 性能のポテンシャルを発揮出来ることを示した．

本稿で述べた分散システムアーキテクチャは，定性的な検討に基づくもので，実装に基づく評価は今後の課題である．また，効率化の尺度として実行時間やシステム負荷だけでなく，電力消費量についても着目し研究を行っていく予定である．

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) の委託事業による成果である．

文　献

[1] J. F. Gantz, C. Chute, A. Manfrediz, S. Minton, D. Reinsel, W. Schlichting and A. Toncheva: “The diverse and exploding digital universe”, An IDC White Paper (2008).

[2] E. Anderson and J. Tucek: “Efficiency matters!”, Proceedings of the SOSP Workshop on Hot Topics in Storage and File Systems (HotStorage09), Big Sky, MT, USA, ACM SIGOPS (2009).

[3] A. Bialecki, M. Cafarella, D. Cutting, O.：“Hadoop: a framework for running applications on large clusters built of commodity hardware”, Wiki at http://lucene. apache. org/hadoop (2005).

[4] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall and W. Vogels: “Dynamo: amazon’s highly available key-value store”, SIGOPS Oper. Syst. Rev., **41**, 6, pp. 205–220 (2007).

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes and R. E. Gruber: “Bigtable: a distributed storage system for structured data”, OSDI ’06: Proc. of the 7th USENIX Symposium on Operating Systems Design and Implementation, pp. 15–15 (2006).

[6] Q. Zhu, Z. Chen, L. Tan, Y. Zhou, K. Keeton and J. Wilkes: “Hibernator: helping disk arrays sleep through the winter”, SIGOPS Oper. Syst. Rev., **39**, 5, pp. 177–190 (2005).

[7] J. Leverich and C. Kozyrakis: “On the Energy (In) efficiency of Hadoop Clusters”, HotPower ’09: Workshop on Power Aware Computing and Systems, Big Sky, MT (2009).

[8] 合田, Q. Wenyu, 喜連川：“”複数問合せ処理を意識したディスクストレージ省電力化に関する一考察””, DEWS2008: 第 19 回データ工学ワークショップ (2008).

[9] H. Yokota: “Autonomous Disks for Advanced Database Applications”, Proc. of International Symposium on Database Applications in Non-Traditional Environments (DANTE’99), pp. 441–448 (1999).

[10] M. Mesnier, G. Ganger and E. Riedel: “Object-based storage”, IEEE Communications Magazine, **41**, 8, pp. 84–90 (2003).

[11] D. Narayanan, A. Donnelly and A. Rowstron: “Write off-loading: Practical power management for enterprise storage”, Trans. Storage, **4**, 3, pp. 1–23 (2008).

[12] J. Dean and S. Ghemawat: “MapReduce: simplified data processing on large clusters”, OSDI’04: Proc. of the 6th conference on Symposium on Opearting Systems Design & Implementation, pp. 10–10 (2004).

[13] Amazon-Inc.: “Amazon Elastic Compute Cloud (Amazon EC2)” (2008).

[14] Z. Li, Z. Chen and Y. Zhou: “Mining block correlations to improve storage performance”, ACM Trans. Storage, **1**, 2, pp. 213–245 (2005).

[15] G. Soundararajan, M. Mihailescu and C. Amza: “Context-aware prefetching at the storage server”, USENIX 2008 Annual Technical Conference, pp. 377–390 (2008).

[16] M. Sivathanu, V. Prabhakaran, F. I. Popovici, T. E. Denehy, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau: “Semantically-smart disk systems”, Proc. of the 2nd USENIX Conference on File and Storage Technologies (FAST03), pp. 73–88 (2003).

[17] M. Sivathanu, L. N. Bairavasundaram, A. C. Arpaci-Dusseau and R. H. Arpaci-Dusseau: “Database-aware semantically-smart storage”, Proc. of the 4th USENIX Conference on File and Storage Technologies (FAST05), pp. 18–18 (2005).

[18] X. Ding, S. Jiang, F. Chen, K. Davis and X. Zhang: “Diskseen: exploiting disk layout and access history to enhance i/o prefetch”, USENIX 2007 Annual Technical Conference, pp. 1–14 (2007).

[19] W. W. Hsu, A. J. Smith and H. C. Young: “The automatic improvement of locality in storage systems”, ACM Trans. Comput. Syst., **23**, 4, pp. 424–473 (2005).

[20] E. V. Carrera, E. Pinheiro and R. Bianchini: “Conserving disk energy in network servers”, Proc. of the 17th intl. conf. on Supercomputing (ICS), pp. 86–97 (2003).

[21] A. E. Papathanasiou and M. L. Scott: “Energy efficient prefetching and caching”, USENIX 2004 Annual Technical Conference, pp. 22–22 (2004).

[22] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz and I. Stoica: “Improving mapreduce performance in heterogeneous environments”, USENIX Symposium on Operating Systems Design and Implementation (OSDI) (2008).

[23] 須田：“ヘテロ並列計算環境のためのタスクスケジューリング手法のサーベイ”, 情報処理学会論文誌. コンピューティングシステム, **47**, 18, pp. 92–114 (2006).

[24] 長坂, 丸山, 額田, 遠藤, 松岡：“Gpu における性能と消費電力 の相関性の解析”, 情報処理学会研究報告 2009-HPC-121, No. 26 (2009-07).

[25] 浜野, 遠藤, 松岡：“ヘテロ並列環境のための省電力タスクスケジューリング”, 電子情報通信学会技術研究報告. CPSY, コンピュータシステム, **108**, 180, pp. 97–102 (20080729).