特集号
招待論文

「社会の頭脳システム」におけるドコモのHadoopクラスタの活用事例

趙 晩熙[†1]　國頭 吾郎[†1]　土橋 昌[†2]　吉田 耕陽[†2]　川崎 紀宏[†1]　鈴木 亮平[†1]
石田 創[†1]　石井健司[†1]　甲本 健[†3]　大町 正史[†3]　遠藤 賢[†4]　田中 聡[†1]

[†1] (株) NTTドコモ　[†2] (株) NTTデータ　[†3] ドコモ・テクノロジ（株）　[†4] NTTソフトウェア（株）

ドコモが取り組む，大規模データ解析から新たなサービスを開発，実行する「社会の頭脳システム」において，最初の大規模データ処理基盤である，1,000台超のサーバからなるHadoopクラスタの構築と運用について述べる．本クラスタは，サーバとネットワークの冗長化により高可用性を実現した．また，コモディティなハードウェアとオープンソースソフトウェアを活用した保守運用により，低コストでの構築と少人数での運用を可能にした．本論文では，ドコモにおけるHadoopクラスタの構築，4年間にわたる運用，新規クラスタへの移行におけるノウハウとそこから得られた知見について述べる．

1. はじめに

ドコモでは，大規模データを処理し，その結果をもとに新たなサービスを開発，実行するためのプラットフォームの研究開発を行っている．これを「社会の頭脳システム」[1]と呼び，2008年頃から検討を始めた．社会の頭脳システムは，大規模データの処理を行うための基盤とサービスを実行制御するための基盤とから構成される．社会の頭脳システムで実現した新サービスの1つが，携帯電話サービスで用いる運用データを使って，時々刻々と変化する人口の推移を推計する「モバイル空間統計」[2]である．モバイル空間統計は，広域なエリアの人口を，時間単位で継続的に推計する．それによって社会の発展・高度化に寄与することを目的としている．

社会の頭脳システムの大規模データの処理を行う基盤の構築にあたり，扱うデータ量がペタバイト級になると想定した[3]．当時，ペタバイト級の大規模データを処理する基盤技術として注目されていたのがHadoop[4]である．Hadoopは，コモディティなハードウェアで構築可能な大規模データ処理基盤のためのオープンソースソフトウェア（以下，OSS）で，Yahoo! をはじめ，Amazon，Facebook，LinkedIn等でも採用されていた[5]．Hadoopは，複数のコモディティなサーバにより構成されたスケールアウトが容易なクラスタの上に，Hadoop分散ファイルシステム（以下，HDFS）による大規模データの蓄積機能と，MapReduce（以下，MR）による大規模データ処理機能を実装している．Hadoopクラスタは，ネームノード（以下，NN）とジョブトラッカ（以下，JT）とスレーブノード（以下，SN）と呼ばれるサーバから構成される．Hadoopクラスタは，SNを追加することにより簡単にHDFSの容量を増やすことができ，SNが多少故障しても，データロスを防ぎ，データ処理の継続が可能なため，保守運用のコストを少なくすることが可能である．我々は，Hadoopの有用性・将来性に期待し，社会の頭脳システムの大規模データ処理基盤としてHadoopを活用することにした．

本論文では，ドコモで初めて本格的に構築した大規模Hadoopクラスタにおける課題と解決方法について述べる．さらに，後継クラスタを構築したことによる，クラスタ間での移行に関する経験についても述べる．

2. 社会の頭脳システムにおけるHadoopクラスタの構築

2.1 システムの概要

社会の頭脳システムにおける最初の大規模データ処理基盤をSteviaと呼ぶ．Steviaは，1,012台のSNを持つHadoopクラスタと，それを監視・運用するためのサーバ群から構成されている．Steviaの構成の概略を**図1**に示す．

SteviaのHadoopクラスタを構成するサーバの主な仕様を**表1**に示す．

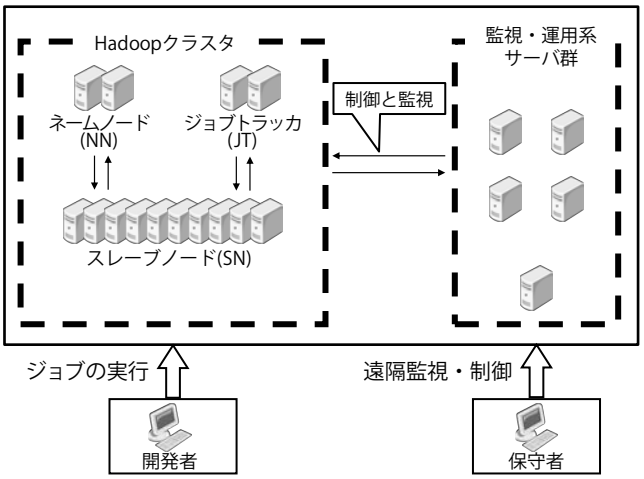

図 1　Stevia のサーバ構成

表 1　Stevia の Hadoop クラスタのサーバ仕様

種別	SN	NN	JT
台数	1,012	2	2
コア数	4	4	4
メモリ容量	8GB	32GB (64GBまで拡張)	8GB (64GBまで拡張)
HDD 容量	1TB × 2	1TB × 2	1TB × 2

Stevia の Hadoop クラスタは，当時としては日本最大級[☆1] の Hadoop クラスタで，総容量 2PB を，1 台あたり 2TB のハードディスクを実装した SN 1,012 台で実現した．Stevia は，Hadoop のバージョン 0.19.2 を使用し，2013 年に新たな大規模データ処理基盤（以下，Walnut）（第 4 章参照）を構築した後も，同一のバージョンで，引き続きサービスの開発のため稼働中である．Stevia は以下の使い方を想定していた．

- 同時に多数の開発者が使用する．
- 解析結果の年単位の比較のため，長期間運用する．
- 数人規模の保守者による保守運用を行う．
- 保守者が常駐する場所とは異なる遠隔地に設置する．そのため，現地の常駐保守者は不在で，1 ～ 2 時間以内の駆けつけはできない．

以上のことから，以下のような開発コンセプトの下で構築，運用を行った．

- **可用性の確保**；Stevia は，多数の開発者により常時使用されるため，Hadoop クラスタの可用性を確保する必要があった．
- **アカウントの認証**；開発者の権限を明確にし，クラスタ上でのアクセス制御を行う必要があった．
- **コストの低減**；システムの規模が大きいことにより，構築のための初期コストはもちろん，電気代や人件費等の継続的な運用コストも高くなるため，できるだけコストを低く抑える必要があった．
- **遠隔監視・制御**；保守者が遠隔でシステムを監視し，容易に制御することを前提にシステムを構築，運用する必要があった．

次節以降で，上記の開発コンセプトの具現化における課題とその解決方法について述べる．

[☆1] OSCON 2007 で発表された Yahoo! の最も大きいクラスタ [6] の規模は 1,600 台であった．

2.2　可用性の確保

2.2.1　サーバの冗長化

予期せぬクラスタの停止を防ぐためには，単一故障点である NN・JT と，監視・運用系サーバ群のサーバの可用性を確保する必要がある．ここでは，NN・JT と監視・運用系サーバ群の高可用性を実現するために Heartbeat[7] と DRBD[8] を用いた，サーバの冗長化について述べる．

NN と JT は耐故障性がなく，その異常や故障はクラスタ全体において持続的に大きな影響を与えかねない．NN と JT の高可用性（以下，HA 化）が実用化されつつあるのは最近のことで，当時は NN と JT の HA 化のための，確立された方法はなかった．そこで，NN・JT の HA 化を実現する方法を独自に実装する必要があった．Stevia では，2 台のサーバ間で Heartbeat と DRBD を活用して NN を冗長化することにした．

図 2 に NN の HA 化構成の概略を示す．

NN 用サーバ（NN1 と NN2）は 2 枚の NIC を持ち，各 NIC のネットワークポートがボンディングされ，異なる 2 台のスイッチを介して接続される．このとき，NN1 と NN2 で動作する Heartbeat により，NN1 と NN2 の動作状態が監視される．仮に NN1 が active な NN で，NN2 が standby な NN とする．NN1 に異常が発生すれば，NN2 がそれを検知して NN1 のプロセスを停止し，自ら active になることで NN の HA 化を行う．NN2 が正常に起動するためには NN2 も HDFS の最新のメタデータ（fsimage と edits ファイル）を保持する必要がある．そこで，NN1

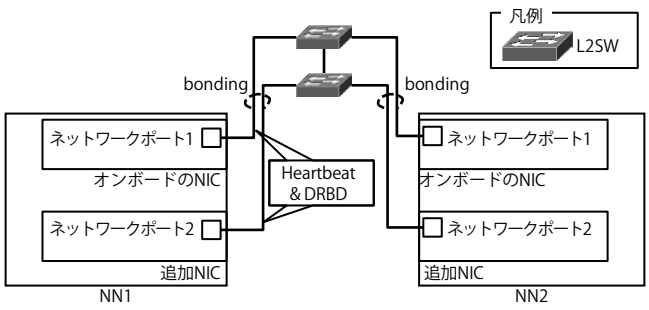

図 2　NN の HA 化

とNN2の間のDRBDによりメタデータを同期させることで，NNの切り替えにおいてもメタデータを失わないようにした．最新のHDFSのメタデータをNN1とNN2で共有するには，共有ディスクを利用する方法も考えられた．しかし，最大でも20GB程度と予想されたメタデータのために共有ディスクを使うのはハードウェアのコストの観点から適切ではないと判断したため，DRBDを活用することにした．

冗長化における副作用が，2台のNNが同時にactiveになるスプリットブレイン問題である．STONITH [9]によってスプリットブレインの状態を回避できるが，当時はSTONITHに関する情報も実績もまだ十分ではなかった．そのため，NWの接続を以下のように工夫することでスプリットブレインの発生を極力防止した．

- 各NNはオンボードのNIC以外に追加のNICを搭載し，1枚のNICが故障してもネットワークが到達できるようにした．
- 各NNを2台の異なるL2SWと接続することで，1台のL2SWが故障してもネットワークが到達できるようにした．

各装置の冗長化により，スイッチの同時故障，同一NN上のNICの同時故障の確率を低くすることができ，スプリットブレインの発生の確率をも十分低く抑えることができると考えた．

JTもHadoopにおける単一故障点であり，異常や故障が発生するとMRプログラムの実行ができなくなるため，図2のNNと同じ方法でJT2台を2台のスイッチに接続した．JTは2台のサーバ間で特に同期すべきデータを持っていないため，Heartbeatだけを使って死活監視と自動切り替えができるようにした．

監視・運用系サーバ群の監視機能を担うサーバは，全サーバからリソース情報と異常情報を受信しながら，Steviaの状態をモニタリングする．この監視機能を担うサーバに異常が発生すると，システム全体のモニタリングができなくなるため，このサーバも単一故障点である．そこで，このサーバも同様の方法で冗長構成にし，2台のサーバ間でリソース情報と異常情報の同期をとるためDRBDを活用し，Heartbeatによって死活監視をすることによってHA化した．

2.2.2 ネットワークの冗長化

Steviaでは，**図3**に示すように，ネットワークも冗長化した．

HadoopはSN間で大量のトラフィックが流れると考えられ，まず，Hadoopの処理に利用するネットワーク（以下，業務系NW）と，監視・運用に使うネットワーク（以下，監視系NW）を分離してお互いに及ぼす影響を減らした．

図3を基に，業務系NWの冗長化構成について述べる．SN23台を1つの単位として，各SNの2つの1Gbpsの通信ポートを2台のL2SW（0系と1系）へそれぞれ接続する．2台のL2SWは同じ46台のSNに接続しており，L2SW同士も2つの1Gbpsの通信ポートで接続している．0系のL2SWは上位の0系のL3SWと接続し，その間は10Gbpsインタフェースを使っている．同様に，1系のL2SWも上位の1系のL3SWに接続している．0系と1系のL3SW同士も10Gbpsで接続されている．これにより，L2SWのポートを極力使い切る構成とした．さらに，SN間の通信ポートは1Gbpsであるが，L2SWよりも上位では10Gbpsでの接続とし，高価な10Gbpsインタフェースの数を少なくすることとした．

図3のL2SWと上位のL3SWの冗長化構成は，当初複数のリンクを使ったたすき型（**図4**（a））で検討を進め

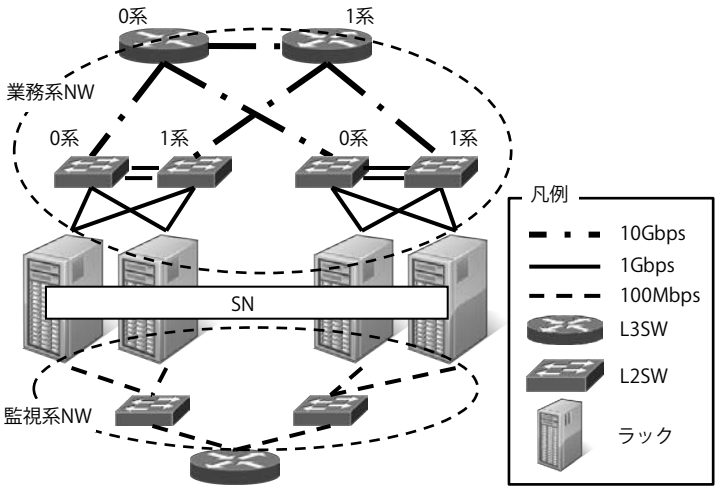

図3　Stevia のネットワーク構成

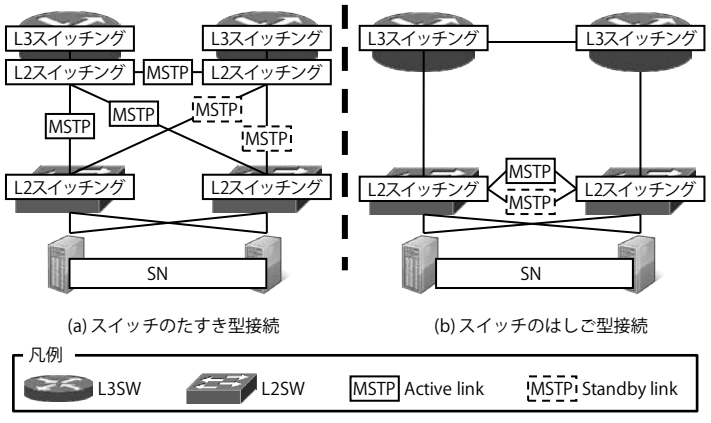

図4　業務系 NW のスイッチの冗長化構成

た．複数のリンクを使った方がリンクの障害に対する耐故障性が高いからである．L3SWとL2SW間ではMultiple Spanning Tree Protocol（MSTP）を使ってスイッチの故障時にもネットワークの到達可能性を確保しようとした．図4のMSTPと表記された四角は，実線がMSTPによりactiveになったリンクを，点線がループを作らないようにMSTPによりstandbyになったリンクを示す．しかし，この構成はSteviaでは動作上問題があった．その問題について述べる．

SteviaではVLANを使ってネットワークを構成している．MSTPはスパニングツリーを計算する際，すべてのスイッチが同じVLANに属していることを前提とする．VLANが異なると，MSTPではなくSTPが動作する．STPはスパニングツリーを計算する際，VLANを無視して1つのスパニングツリーを作り，複数のVLANを考慮しない．そのため，複数のVLANに属するあるリンクの故障によりスパニングツリーを再計算する場合，あるVLAN上では正常なスパニングツリーが作られても，別のVLAN上では作られなくなるケースが発生してしまう．スパニングツリーが作られないVLANでは，ネットワークの到達性が失われる．そのため，図4(b）のようなはしご型の構成にし，さらにL2SW間の配線を冗長化している．こうすることで1台のL2SW，または，1台のL3SWが故障してもネットワークの到達可能性が保証される．

監視系NWは特に冗長化は行わず，各サーバと監視系NWのスイッチを単純なツリー構造のネットワークにした．

2.3　アカウントの認証

個々の開発者の権限を明確にし，それに基づいたクラスタ上のアクセス制御を実現するため，Steviaへのログインサーバを設け，開発者，保守者のアカウントを認証し，ログイン履歴と操作履歴を記録した．Steviaにアクセスする開発者，保守者は必ずログインサーバのOS上のアカウントで認証を行うようにした．

図5にSteviaへのアクセス経路の概略を示す．ログインサーバにはSteviaにアクセスする開発者，保守者のOS上のアカウントを設定し，それによってログイン認証を行う．ログインサーバへログイン後，Hadoopクラスタへアクセスする開発者はさらにクライアントのOS上のアカウントでクライアントへログインし，監視・運用系サーバへアクセスする保守者は監視・運用系の各サーバのOS上のアカウントでログインする．

2.4　コストの低減

2.4.1　初期コストの低減

コモディティなハードウェアによりシステムを構築することで，初期コストの低減を図った．

SteviaのHadoopクラスタは，表1に示したように，コストパフォーマンスが優れたコモディティサーバを使って構築した．監視・運用系サーバも，モデル番号で仕様が特定できるような一般的な製品を使用した．

図3のL3SWはネットワーク帯域の確保のため高速で高価なものを採用している．しかし，L2SWは48ポートを持つコモディティな装置を使い最大限のSNを1つのL2SWに接続することでポートの無駄をなくした．これで，L2SWの台数を減らし，上位のL3SWとの10Gbpsでの接続数を減らす構成とした．これにより，高価な10Gbpsの光ポート数を減らすことができ，スイッチのコストを抑えることができた．このように，コモディティサーバとコモディティスイッチを集めてクラスタを構築することで初期コストを低減した．

2.4.2　少人数による監視・運用のためのOSSの活用

少人数で効果的に監視・運用をするためにSteviaではKickstart[10]，Ganglia[11]，Nagios[12]，Cacti[13]等OSSのツールを活用した．

Kickstartは，OSや各種ソフトウェアのインストールを自動化するツールである．Gangliaは，CPU，メモリ，ディスク等リソースの状態の変化を短時間で観測できるようにした．Hadoopには，各サーバのHadoopプロセスを監視することでHDFSの状態やジョブの進捗状況をモニタリングする仕組みが備わっているが，それだけではOSレベルでの異常を適切に検知できないためGangliaを活用した．Cactiは，ネットワークトラフィック情報を

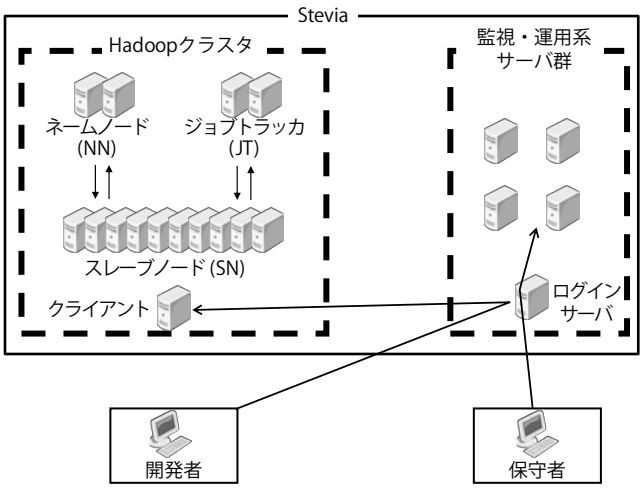

図5　ログインサーバによる認証

収集することでネットワーク状態のモニタリングを可能にした．Hadoopではジョブの処理中にSN間で発生するデータのやりとりによりネットワークがボトルネックになると想定したため，Cactiを導入した．さらに，Nagiosはサービスや機能が問題なく動いているかを監視し，異常状態を通知してくれるツールである．

各種ツールを使うことで，少人数で，容易にシステムを構築したり，HDFS利用量の動向を把握して増設を検討したりすることができた．また，リソースの利用状況を確認してプロセスダウンの原因を追跡したり，ネットワークのトラフィック量を測定してネットワーク設計の妥当性を判断することもできた．

2.5　遠隔監視・制御

異常が確認されたサーバは必要に応じて遠隔で操作する必要がある．遠隔でプロセスを再起動したり，OSを再起動したり，電源を入れ直したりするため，すべてのサーバに，遠隔監視・制御を可能にするボードを搭載した．これにより，各サーバのディスク構成を変更するなど，OSレベルでの再構成が必要になる場合でも遠隔でのPXEブートによりOSを再インストールすることができた．さらに，コンソールにしか現れない情報も確認することができるようになった．それでも解決できないようなハードウェアの障害の場合は，現地での作業で対応するような体制を作った．

3. 社会の頭脳システムの運用で得られた知見

Steviaを1ビットのデータロスもなく4年間運用してきた．不安定だったのはSteviaを構築してから最初の約1年間で，障害による停止は10回あった．ここでは，HDFSの操作が可能かどうかにかかわらず，ジョブが実行できない状態を停止と定義した．停止の原因はすべてNNとJTのメモリ枯渇によるものであった．そのため，NNとJTのメモリを増設することでその後の稼働率は99.5%まで改善することができた．その後も2014年1月の現在まで常時稼働している．

SNの障害発生時には1カ月あたり0.5人程度の稼働で障害解析を行った．システムの全停止時を除けば，4年間で稼働しているSNの台数が1,000を下回ったことは一度もない．コモディティサーバであるにもかかわらず，現地で対応しなければならない故障も，月に1回の現地作業で対応できる程度であった．

ここでは，第2章で述べた開発コンセプトの具現化に関する知見，および，運用の経験を通して習得したいくつかのノウハウについて述べる．

3.1　NNとJTの冗長化の重要性

Heartbeat や DRBD 等を使って単一故障点のサーバを冗長化し可用性を高めることが必要である

Heartbeatを使ってNNの状態をモニタリングすることによって，NNの異常を自動で，早期に検出することができた．また，DRBDを使ってNNの最新のメタデータを同期することによって，データロスを防ぐことができた．

実際の運用期間中には，現用系のNNに障害が発生したことはなかったが，メンテナンスのための手動切り替え時に確認された本方式の課題として，切り替えの遅延が挙げられる．NNの切り替え時間はfsimageが5GB程度の場合約30分かかった．NN起動時に，fsimageをメモリにロードし，SNからのブロックレポートを受信しながらHDFSの健康状態を確認する処理に時間がかかるためである．これは，自動切り替えだけではなく，通常のNNの起動でも同じ時間がかかり，fsimageが大きくなればなるほど切り替えと起動の時間は長くなる．

JTも，Heartbeatを使用しているが，NNとは異なり，もともと起動にそれほど時間がかからないため，約1～2分で自動切り替えが完了した．

異常による自動切り替えはJTに4年間で1回程度しかなかったが，HA化のためにさまざまな起こり得る障害を想定した試験を通じてリスクを把握することで，現実的な運用方式・体制を整理できた．

3.2　監視・運用のための OSS の活用

扱うデータ量や，計算量が変化し続けるシステムにおいては，リソース利用状況を正確に把握し，変化を記録，追跡できる仕組みが必要である

利用状況を把握するためのツールの一例として，Gangliaについて述べる．

図6は，Gangliaにより計測し可視化した初期の1年間のNNのメモリ使用量を例として示している．横軸が時間軸で，縦軸は使用メモリ量[GB]を示す．メモリ使用量

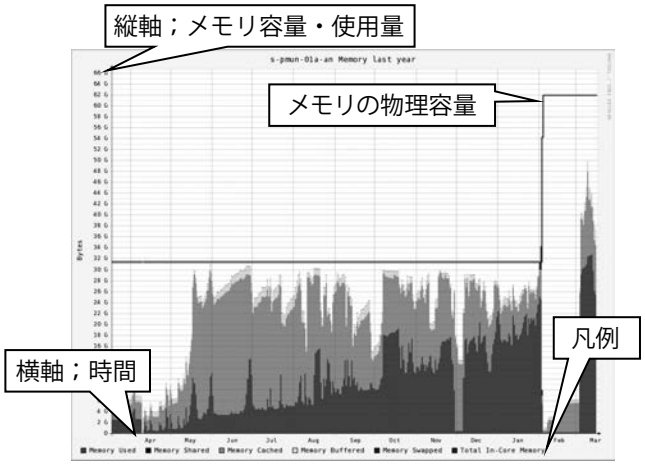

図 6　Ganglia によるメモリ使用量のモニタリングの例

のグラフではメモリの物理容量が横太線で示される．濃い部分が実際の使用メモリ量であり，時間とともにメモリ使用量が増えていることが分かる．Ganglia を使ってモニタリングした結果からメモリ使用量の傾向を確認することが可能になり，これに基づき，NN のメモリが不足する前にメモリ増設等の対策を講じることができた．また Ganglia では，メモリ使用量以外にもディスクの使用量や CPU の使用率等についてもモニタリングすることができる．Hadoop は計算結果を蓄積するストレージの側面と，計算を実行する計算リソースの側面があり，それぞれの影響が顕著に表れるリソースを個別に観測できた．また Stevia で採用したバージョンの Hadoop では，HDFS 全体の利用状況を Ganglia で可視化する仕組みがなかったため，それを独自に追加開発した．特に Ganglia は長期間のデータを可視化することが得意なため，長期的な Hadoop 利用傾向を把握するのに有用であった．

大規模システムで Ganglia を使うときの大きな問題点は，リソース情報のディスクへの書き込みが大量に発生することである．Ganglia はデフォルトで 15 秒に 1 回データを計測しディスクへ書き込む．そのディスクへの IO がボトルネックとなり，モニタリング機能が正常に動作しなかった．そこで，Stevia では計測する時間間隔を 1 分に延ばし，ディスクへの書き込みに ramdisk を利用することにより IO のボトルネックを緩和した．

Kickstart は，大規模な台数に対してシステムのインストールを簡単にするツールとしてたいへん有用であった. 同時に複数台に対してのインストールができたため，構築にかかる時間も短縮できた．ただ，Stevia は，利用傾向に応じて幾度も構成変更されており，Kickstart だけでは設定変更の追随が難しかった．そのため，Walnut においては，構成管理ツール Puppet[14] を導入した．

Nagios は全サーバのプロセスの異常やログの中の異常メッセージを発見するために導入した．Nagios は，設定ファイルがテキストファイルであるため，設定ファイルをデータベース内に管理するものに比べユーザの可読性が高く，テキスト形式の設定ファイルを機械的に管理，変更する Puppet と相性が良い．

Nagios を使うときの注意点として，あるイベントが発生したことを示すログが膨大になり，それを逐次監視するのが困難になるということがある．そこで，イベントを発見した際に状態情報を含めて記録するよう，監視スクリプトを工夫することで，監視すべきログの量を調整した．

Hadoop では SN 間で大量のデータのやりとりが発生すると想定した．そこで，ネットワークの状態をモニタリングするため Cacti を活用した．これによりネットワーク帯域に対する実際のトラフィックの量を容易に確認できた．そのようにネットワークトラフィックの傾向を把握することができるため有用であった．

以上のように，Stevia で活用したツールはすべて有用であった．後継のクラスタでも Kickstart，Ganglia，Nagios，Cacti に加え，Puppet を活用することで設定変更に柔軟に追随することができた．

3.3　サーバのメモリ量の設計

サーバのメモリ設計では，データの属性，ジョブの数，タスクのメモリ要求量，コア数を考慮することが重要である

Stevia の常時運用に伴い，多数のジョブが継続して投入されるようになった．そのため，Ganglia でモニタリングした NN と JT はメモリ使用量が高く当初の容量（NN と JT がそれぞれ 32GB と 8GB）では不十分であり，スワップが発生する事態が頻発した．

NN のメモリ使用量の増加は，NN が管理しなければならないメタデータが増加したことに起因し，さらにこれは HDFS が格納する処理結果のファイル数とファイルサイズに起因する．生成されるファイル数が多いクラスタほど，小さなファイルが多く生成されるクラスタほど，メモリ容量を大きくする．その際のメタデータ量の見積りは，ファイルごと，ブロックごとに最大 200 バイト使うという仮定で計算できる [15]．実際 Stevia で計算した結果 200 バイトよりは小さい値だったが，マージンを考えて 200 バイトで見積もっても差し支えない．

JT のメモリ使用量の増加は，JT が保持するジョブの

履歴や状態の量が増加していることに起因する．ジョブによってJTのメモリ使用量が異なるため，事前にGangliaなどを使ってメモリ使用量を確認しておくことも有効である．

SNのメモリの使用量に関しては，SN上で実行されるタスク数と，各タスクを実行するchildJVMに割り当てる最大ヒープサイズに依存する．SN上で実行されるタスク数は，設計時にコア数を考慮しながら設定する．当初Steviaでは，**図7**のようにSNの全体の容量8GBに対して，OSのプロセスが約1.5GB，データノードプロセス（以下，DN）が1GB，タスクトラッカプロセス（以下，TT）が1GBと想定した．そのため，SN1台あたり残りの約4.5GBがタスクの実行に利用できると考えた．さらに，社会の頭脳システムのタスクを実行するためのchildJVMが最大1GBのヒープであれば十分と想定したため，残りの約4.5GBをchildJVMに割り当てる場合，4つのコアがそれぞれ1GBのメモリを使って1つのタスクを実行できると考えた（図7(a)）．しかし，実際に実行するMRプログラムが，最大ヒープサイズが1GBや1.5GBの場合ではメモリ不足により失敗した．メモリ不足の原因の1つとしては，想定よりも処理内容が複雑であり，データ種類の多かったことが考えられる．たとえば，メモリ上にデータを蓄積しながら動作したり，大きなデータをオンメモリで結合するような処理を行う場合，メモリ使用量が大きくなる要因になり得る．最終的にSteviaでは，図7(b)のように，1つのタスクに割り当てる最大ヒープサイズをそれより大きい2GBにした．その結果，SNのCPUコア数が4であるにもかかわらず，同時実行タスク数を2に減らさざるを得なかった．必要とされる最大ヒープサイズが2GBの場合，同時に実行されるタスク数をコア数分まで増やすためには最低でも，（DNのためのメモリ量）+（TTのためのメモリ量）+（OSのためのメモリ量）+（2GB × コア数）＝11.5GBのメモリ容量が必要だった．

Hadoopのプラクティカルなハードウェア構成は，コモディティサーバの仕様でよいとされている．しかし，実際の運用から，クラスタの規模にかかわらずサーバ1台の仕様，特にメモリの容量はかなり重要であることが分かった．以上で述べたように，NNのメモリ容量はクラスタに格納されるファイルの大きさに関する属性を考慮して見積もるべきである．JTは，実行されるジョブの数と，各ジョブが必要とするメモリ使用量を考慮に入れてメモリ容量を見積もる必要がある．SNは実行されるタスクのメモリ要求量と，同時実行されるタスク数を基にメモリ容量を決めるべきである．

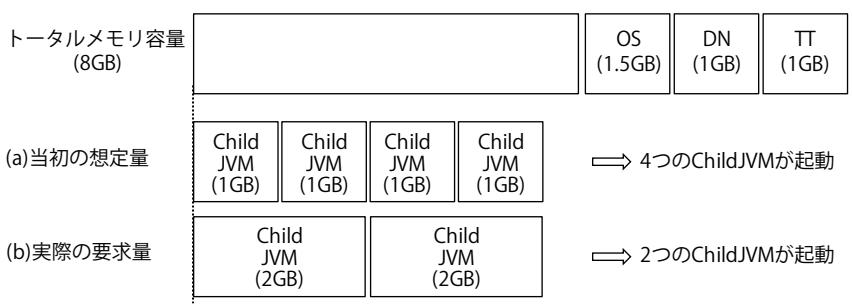

図7　SNのメモリの割り当て

3.4　HDFSの可用容量の確保

サーバ追加によるスケールアウトに加えて，ファイル形式の選択や圧縮によりデータを適切な形で保存することで，HDFSのストレージとしての費用対効果を高めることが重要である

Steviaを長期間運用すると，HDFSの容量が足りなくなってきた．HDFSの容量を確保するための対策として，大きく以下の2つの方法を実施した．

- HDFSの総容量の追加
- HDFSの使用量の低減

Hadoopの特長の1つとして，SNの追加によるスケーラビリティが挙げられる．Steviaでも別途138台のSNを追加し，約200TBの容量を増設した．あわせて，HDFSの使用量を減らすため，MRの中間ファイルのような処理後不要になるデータは，レプリカの数をHadoopで推奨されている3から2に減らすことでHDFSの使用量を節約した．さらに，可能な限りHDFSに入れるファイルは圧縮した．Steviaのデータは圧縮によってそのサイズが約1/10になったため，効果が大きい．しかしながら，通常のファイルをgzip圧縮すると，複数のブロックに分割されたファイルについては，MR処理の際全ブロックを1台のSNに集めないと解凍できなくなる．そこで，ファイル形式をHadoop独自の形式であるsequence fileにしてgzip圧縮することで，上記の問題を解決した．

一般的に，HDFSの容量設計では，処理によるデータの大きさと，保存期間等を十分検討する必要がある．しかし正確なデータ量の見

積もりは難しく，容量枯渇への対策が必要となる場合も多い．

スケールアウトによる容量追加に加えて，結果データの他ストレージへの退避や適切な圧縮処理，レプリケーション数の調整により，ストレージの費用対効果を高めることが重要である．

3.5　HDFS 上のデータのファイルサイズ

ディレクトリ構造の非階層化によるファイル数削減や，データサイズに応じた MR 処理時の並列度の設定を行い，HDFS 上の小さなサイズのファイルの増加を防止することが重要である

Hadoopは，数十MB以上の大きなファイルを効率的に扱うように作られている．しかし，Steviaではブロックサイズよりも小さなファイルがたくさん生成された．その理由として以下の2つの点が挙げられる．

- 業務ロジックを優先したディレクトリの設計
- Hive[18]によるジョブ実行時の過度な分散度

それぞれについて述べる．

Steviaでは，ディレクトリを設計する際，業務ロジックを優先したディレクトリ構造にした．その結果，データが過度に細分化され，ブロックサイズにも満たない小さなファイルがたくさん生成される結果となった．

さらに，MR処理のReduce数を増やしたことが原因の1つである．MRの出力ファイルはReduce数に依存する．Reduce数を増やすことは大規模データ処理のスループット向上に寄与する．しかし，小さくなった入力データに対しても同様のReduce数を指定したため，出力ファイルが過度に細分化されることとなった．

小さなファイルによって必要以上のメタデータが生成されると，NNのメモリを圧迫する．そのため，データが細分化されないディレクトリ構造を設計し，入出力データのサイズを考慮し必要以上の並列度を設定しないことで小さなファイルの生成を減らすべきである．

4. クラスタ間のマイグレーション

Walnutは，Steviaより大きな容量を少ない台数や小さい設置面積で実現し，Steviaとは異なる遠隔地に構築したクラスタである．Walnutは，Steviaとは異なるバージョンのHadoopをインストールした．本章ではSteviaからWalnutへのマイグレーションに関する検討について述べる．

4.1　データの移行

クラスタ間のデータ移行の検証を行った．本検証では，最悪なケースの遅延を見積もるため，Hadoopが最も苦手とする種類のテストデータを生成し，実際に移行を行った．

検証に用いたデータの属性と環境を示す．

- データの属性
  - ➢総データサイズ：65GB
  - ➢総ファイル数：約168万個
  - ➢ディレクトリ数：7個
  - ➢平均ブロックサイズ：48KB
- 環境
  - ➢Hadoopのバージョン：Stevia : 0.19.2，Walnut : 0.20.2
  - ➢SNの数：Stevia : 1,012，Walnut : 210
  - ➢SteviaとWalnut間のネットワーク帯域：440Mbps

検証した以下の3つの方法は，クラスタ間でデータを移行する際に容易に考えられる方法である．

- 方法1．distcp ; hadoop distcpコマンドを使って並列にデータをコピーする方法
- 方法2．dfs -get ; HDFSのファイルをクライアントのディスクにコピーし，scpでWalnutのクライアントにコピーした後，再度HDFSに格納する方法
- 方法3．archive & distcp ; データをHadoop archive（以下，har）ファイルに変換し，それをhadoop distcpコマンドで並列にコピーした後，元に戻す方法

各方法による所要時間を**表2**に示す．

方法1は，ネットワークの帯域は十分余裕があるにもかかわらず，所要時間は6時間34分であった．

方法2は，「hadoop dfs -get」コマンドによってHDFSからクライアントにデータをダウンロードする処理だけで13時間10分を要した．あまりにも所要時間が長かったため，Walnutへの格納は実施していない．

方法3では，7つのディレクトリにあるファイルをディレクトリ単位でhar化した．har化は同時実行が可能である．このとき，7つのhar化処理を同時に実施し，最長のhar化処理に2時間29分を要した．その後，「hadoop

表 2　移行の所要時間の比較

移行の方法	所要時間
方法 1．distcp	6 時間 34 分
方法 2．dfs –get	13 時間 10 分
方法 3．archive & distcp	5 時間 23 分

distcp」コマンドを使って7つのharファイルの並列コピーに25分を要した．har化した7つのファイルを元のディレクトリ構造に戻すにはhar化処理と同程度の時間が必要で，最長で2時間29分かかった．よって，方法3が移行の所要時間を最も短縮できた．

以上のように，distcpは同じサイズのデータをコピーするにも，ファイルの大きさにより容易に実行時間に影響する．さらに，小さいファイルがたくさん含まれる場合には，ファイルを適切な大きさに集約（たとえば，har化）してから並列コピーする方法が適切であると考える．

4.2 プログラムの移行の容易性

Steviaでは，HiveQLで記述されたプログラム（以下，Hiveプログラム）と，Javaで記述されたMRプログラムが動作していた．Walnutの構築に伴い，MRプログラムとHiveプログラムも移行する必要があった．このとき，Hiveプログラムは，特に何の変更もしなくてもWalnut上で実行できた．一方，MRプログラムは当然ながらコンパイルしたときのHadoopのライブラリのバージョンが違うため動作しなかった．このとき，再コンパイルだけではなく，Javaプログラムの修正も必要であった．

さらに，HiveQLの実行時間をSteviaとWalnutで比較した．その結果，ほとんどのクエリ文において，Walnutの方がSteviaに比べ2 〜 10倍短かった．しかし，よく使われる特定のクエリ文において，Steviaでは1つのMR処理に変換されるのに対し，Walnutでは2つのMR処理に変換されるケースがあった．このときの実行時間は，Walnutの方がSteviaより1.3 〜 2倍長くかかった．

以上から，Hadoopのバージョンが異なるクラスタ間でのプログラムの移行の容易性に関して，HiveQLによるプログラムの方がJavaによるMapReduceプログラムよりもメリットは高いと考えられる．しかし，実行時間に対するユーザの要求条件が厳しいとき等，場合によってはHiveQLによるプログラムの見直しが必要になる．

5. おわりに

本論文では，2009年にドコモが構築し，4年にわたって社会の頭脳システムに応用した，日本最大級のHadoopクラスタについて述べた．

Steviaの構築にあたっては，可用性の確保，アカウントの認証，コストの低減，遠隔監視・制御の具現化を図った．可用性の確保のためには，サーバとネットワークを冗長化し，アカウントの認証のためにはログインサーバを設けた．コモディティなサーバやネットワーク機器，OSSの監視・運用ツールを活用することでコストを低減することができ，すべてのサーバに遠隔監視・制御ボードを搭載することで遠隔での監視・制御が可能になった．

ネームノードを冗長化して高可用化を実現し，4年間に1ビットのデータロスも発生しなかった．各種オープンソースソフトウェアのツールによって，大規模システムの構築，状態と傾向の把握，および，異常の発見が容易に実現できた．その一方で，MRプログラムが使うメモリの要求量を事前に把握することができなかったためスレーブノードのメモリ容量は十分ではなかった．

これまでの運用を通じ，以下のような知見を得た．同時実行するタスク数を増やすためには，タスクあたりのメモリ要求量を把握し，より多くのメモリ容量をスレーブノードに搭載する必要がある．HDFSの容量を確保するためにはsequence fileを圧縮して格納すべきである．また，HDFSの容量は無制限ではないため，処理する必要があるものだけをHDFSに置くなどの工夫も重要である．Hadoopの特長を考慮すると，少人数の人員でもHadoopを運用することは十分可能である．クラスタ間の移行のためには，Hadoop archiveを用いて積極的にデータをアーカイブすべきである．

最後に，4年間Hadoopクラスタを活用しながら最も重要と考える課題を2つ挙げる．1つは小さいファイルに関する課題であり，もう1つは高速処理に関する課題である．それぞれ現時点と将来の課題である．

Hadoopは大きなサイズのファイルを前提にしている．しかし，実際ユーザがHadoopで処理するファイルがすべて大きなサイズではない．さらに，処理の並列度を上げることによってどうしても小さなファイルがHDFS上に大量に生成されてしまう．Hadoopが小さなサイズのファイルをより効果的に扱える方法があれば，HDFSの利用効率も改善されると考える．

大規模データに対するリアルタイムの処理も課題と考える．Impala[19]やSpark[20]等によってHDFS上のデータに関する高速でインタラクティブな処理が可能になりつつある．しかし，まだ十分な実績がないため，社会の頭脳システムにおける今後の遅延の要求を満たせるか否かの検証をこれから進めていきたい．

参考文献

1) 石田　創, 趙　晩熙, 石井健司, 國頭吾郎, 中山　誠, 鈴木亮平, 越智大介, 川崎紀宏, 大町正史, 甲本　健 : 社会の頭脳システム―モバイル空間統計を支える大規模データ処理基盤 , ドコモテクニカルジャーナル, Vol.20, No.3, pp.24-29 (Oct. 2012).
2) 岡島一郎, 田中　聡, 寺田雅之, 池田大造, 永田智大 : 携帯電話ネットワークからの統計情報を活用した社会・産業の発展支援―モバイル空間統計の概要, ドコモテクニカルジャーナル, Vol.20, No.3, pp.6-10 (Oct. 2012).
3) 堀越　功 : NTT ドコモが巨大マイニング設備構築 , 日経コミュニケーション 2009 年 10 月 1 日号, pp.30-31.
4) Hadoop, http://hadoop.apache.org/ (2008).
5) Hadoop Wiki PoweredBy, http://wiki.apache.org/hadoop/PoweredBy/ (2008).
6) Baldeschwieler, E. : Yahoo! Experience with Hadoop, OSCON 2007, pp.23-27 (July 2007).
7) Heartbeat, http://linux-ha.org/wiki/Main_Page/ (2009).
8) DRBD, http://www.drbd.org/ (2009).
9) STONITH, http://linux-ha.org/wiki/STONITH/ (2009).
10) Anaconda/Kickstart, http://fedoraproject.org/wiki/Anaconda/Kickstart/ (2009).
11) Ganglia Monitoring System. http://ganglia.info/ (2009).
12) Nagios, http://www.nagios.org/ (2009).
13) Cacti, http://www.cacti.net/ (2009).
14) Puppet, http://puppetlabs.com/ (2010).
15) Konstantin V. Shvachko : HDFS scalability : The Limits to Growth, LOGIN : Vol.35 No.2, pp.6-16 (Apr. 2010).
16) White, T. : Hadoop, The Definitive Guide. O'Reilly (June 2009).
17) Datanode behaves Badly when One Disk is Very Low on Space (2010), https://issues.apache.org/jira/browse/HDFS-788
18) Hive, http://hive.apache.org/ (2010).
19) Cloudera Impala : Real-Time Queries in Apache Hadoop, For Real, http://blog.cloudera.com/blog/2012/10/cloudera-impala-real-time-queries-in-apache-hadoop-for-real/ (2013).
20) Spark Lightning-Fast Cluster Computing, http://spark.incubator.apache.org/ (2013).

趙 晩熙（非会員）jo@nttdocomo.com
2001 年筑波大学大学院工学研究科電子情報工学専攻博士課程修了．博士 (工学)．同年 (株) NTT ドコモ入社．IP モビリティ，ネットワークモビリティ，ビッグデータの処理基盤の研究開発に従事．IEEE 会員.

國頭 吾郎（正会員）kunitou@nttdocomo.co.jp
2000 年東京大学大学院工学系研究科電子情報工学専攻博士課程修了．博士（工学）．同年同大学院新領域創成科学研究科基盤情報学専攻リサーチアソシエイト．2002 年（株）NTT ドコモに入社．ユビキタスコンピューティング，ビッグデータ処理基盤，Web サービスの研究開発に従事．電子情報通信学会，IEEE 各会員.

土橋 昌（非会員）dobashim@nttdata.co.jp
2008 年東北大学大学院情報科学研究科応用情報科学専攻修士課程修了．同年（株）NTT データ入社．Hadoop 等の OSS を利用した開発案件に従事.

吉田 耕陽（非会員）yoshidakuy@nttdata.co.jp
2011 年琉球大学大学院理工学研究科知能情報工学専攻修士課程修了．同年（株）NTT データ入社．Hadoop 等の OSS を利用した開発案件に従事.

川崎 紀宏（非会員）kawasakin@nttdocomo.co.jp
2002 年慶應義塾大学大学院理工学研究科開放環境科学専攻修士課程修了．同年（株）NTT ドコモに入社．ユビキタスコンピューティング，分散処理基盤の研究開発に従事．電子情報通信学会会員.

鈴木 亮平（非会員）ryohei.suzuki.eu@nttdocomo.com
2010 年東京大学大学院情報理工学系研究科電子情報学専攻博士課程修了．博士（情報理工学）．同年（株）NTT ドコモ入社．ユビキタスコンピューティング，分散処理基盤の研究開発に従事．電子情報通信学会会員.

石田 創（非会員）ishidas@nttdocomo.co.jp
1993 年年東北大学大学院工学研究科修士課程修了．同年（株）エヌ・ティ・ティ移動通信網（現 NTT ドコモ）に入社．ローミング，IPv6，ビッグデータの処理基盤の研究開発に従事．現在 NTT ドコモ先進技術研究所に所属.

石井 健司（非会員）ishiike@nttdocomo.co.jp
1993 年九州大学大学院工学研究科電子工学専攻修士課程修了．同年日本電信電話（株）入社．1998 年から（株）NTT ドコモ．IMT-2000 交換機 , 移動通信用 IP ネットワーク，分散処理基盤の研究開発に従事．電子情報通信学会会員 .

甲本 健（非会員）koumoto@docomo-tech.co.jp
2005 年東京農工大学大学院工学研究科物理システム工学専攻修士課程修了．同年ドコモ・テクノロジ（株）に入社．ビッグデータ処理基盤の研究開発および運用に従事.

大町 正史（非会員）oomachi@docomo-tech.co.jp
2006 年電気通信大学電気通信学部情報通信工学科学士課程修了 . 同年ドコモ・テクノロジ（株）入社．ビッグデータ処理基盤の研究開発および運用に従事 .

遠藤 賢（非会員）endou.ken@po.ntts.co.jp
現在 NTT ソフトウェア（株）に所属．ビッグデータ処理基盤の研究開発および運用に従事.

田中 聡（正会員）t.satoshi@nttdocomo.com
1987 年慶應義塾大学大学院理工学研究科管理工学専攻修士課程修了．同年日本電信電話（株）入社．2004 年から（株）NTT ドコモ．ユビキタスコンピューティング，分散処理基盤の研究開発に従事．電子情報通信学会会員.

投稿受付：2013 年 11 月 22 日
編集担当：黒橋禎夫（京都大学）