「マルチメディア，分散，協調とモバイル
(DICOMO2016)シンポジウム」 平成28年7月

隠蔽された分散処理環境を透過的に利用可能な基盤の提案

阿部 博[1,2]　井上 朋哉　篠田 陽一[2]

概要：分散処理システムはリソース管理モデルと処理実行スケジューリングに強く依存しており，アプリケーションソフトウェアを常に高い性能で動作させるためには，ソフトウェア開発者がシステムの構成や内部状態の把握をした上で，分散処理システムを強く意識したソフトウェア開発を行う必要がある．ソフトウェア開発者が，分散処理システムのリソース管理や処理実行スケジューリングを意識せずに，「1 台の計算機で行う処理を，透過的にスケールアウト可能な分散処理システムで実行する基盤」が利用できれば，システムの内部状態を意識しながら行うソフトウェア開発にかかる開発コストは抑えられる．そこで本研究では，「1 台の計算機環境と同等の処理を実行可能なスケールアウトする分散処理環境」「1 台の計算機環境と同等のストレージアクセスを分散ストレージで行うアクセス抽象化」「1 台の計算機環境と同等の処理を分散処理環境で実行可能なスケジューラ」を実現し，システムの内部状態を隠蔽しつつ処理を分散実行する実装を行うことで問題解決を行った．また，概念検証のための実装より有効性の評価を行った結果問題となった処理実行スケジューリングへの解決策として，スケジューラへの DAG アルゴリズムの導入を行い，アクセス抽象化のために用いた FUSE によるアクセス遅延を解消するためのデータの保存場所と処理実行ノードの関係を意識したシステムアーキテクチャを再考した．

Proposal of the tranparently available distributed infrastructure

HIROSHI ABE[1,2]　TOMOYA INOUE　YOICHI SHINODA[2]

1. はじめに

小さな IoT デバイスからデータセンターに設置される計算機やネットワーク機器まで，多くの機器はネットワークを通じて大量の情報を送受信することが可能となった．大量の情報はデータ処理基盤を用いて利用者の用途に応じた情報の抽出や計算が行われる．これら収集された情報に対する抽出や計算は，1 台の計算機では扱いきれない処理規模になることがあり，複数台の計算機資源を利用しなければならない場合には，計算機間を高速なネットワークで結合した分散処理システムが用いられる．

分散処理システムは複数台の計算機が連携した独立システムやクラスタ，ミドルウェアとして動作する．それらのシステムは処理実行スケジューラやリソース管理機構，システム間の接続制御機構からなり，単体の計算機とその上で動作するオペレーティングシステムからなるシステムとは大きく異なる．これによりソフトウェア開発者は利用する分散処理システムと 1 台の計算機からなる開発環境の差異を深く意識する必要があり，分散処理システムにおける開発制約が強い．

分散処理システム上で動作するアプリケーションの性能向上を実現するには，開発者は本来ソフトウェア開発で意識したくない処理粒度や処理間の依存を考慮した処理実行スケジューリングやリソース管理機構の様な開発には直接関係のない内部情報を意識して開発を行う必要がある．開発者はソフトウェア開発に集中するために，性能向上は分散処理システムへと任せ，資源管理や処理実行スケジューリングを気にせず 1 台の計算機と同等に分散処理システムを扱うべきである．

しかし分散処理システムを利用する上では常にシステムの状態を把握つつ開発するソフトウェアにフィードバックをし性能向上を目指さなければならない．ソフトウェア開発者が，分散処理システムのリソース管理や処理実行スケジューリングを意識せずに，「1 台の計算機で行う処理を，透過的にスケールアウト可能な分散処理システムで実行する基盤」が利用できれば，システムの内部状態を意識しな

1　株式会社 IIJ イノベーションインスティテュート
2　北陸先端科学技術大学院大学

がら行うソフトウェア開発にかかる開発コストは抑えられる．

本稿は「1 台の計算機で行う処理を，透過的にスケールアウト可能な分散処理システムで実行する基盤」を実現するために，2 章では関連研究に関する調査と問題点を提示し，3 章では提案手法とシステム構成に関する解説と問題解決の範囲を決定する．4 章ではコンセプト実装の結果を開示し，5 章ではまとめとコンセプト実装から得られた今後の課題について述べる．

2. 関連研究

2.1 分散処理フレームワーク

データセンター内やオンプレミス環境で分散処理を行う基盤としては，HPC(High Performance Computing) で主に用いられる MPI フレームワーク，Apache Hadoop[1] や GPGPU を使った処理基盤など多数の分散処理ソフトウェアが存在する．これらのソフトウェアは複数台の計算機資源をまとめあげ，投入される処理に対して効率的に資源を割り当て高速に実行し結果を得るために利用される．

コモディティサーバを利用して安価に構築可能なシステムとしてオープンソースソフトウェアである Hadoop エコシステムが多く利用される．Hadoop エコシステム上で動作するミドルウェアである Spark[2]，Hive[3]，Impala[4] の様に，分散処理を MapReduce[5] プログラムとして記述するのではなく，ライブラリの利用や SQL として記述するものが多い．しかし，これらのライブラリや SQL の様な記述方式は，ミドルウェアのバージョンが更新されるたびに記述ルールや実行形式が変わることが多く，システムの内部状態を把握する手法も変わるため，更新前の開発スタイルからの変化を追従することが必要となり開発コストが高まる．

また，利用する分散処理システム環境と同等の開発環境をユーザが自身の環境へと構築する場合に，仮想マシンやコンテナの様な擬似環境を使うことで実現可能であるが，実祭の分散処理システムの様に複数台の計算機の内部状況を正しく把握することはできず，開発したソフトウェアが実環境でソフトウェア開発者が意図した通り動作する保証はない．

2.2 分散ストレージ

分散処理システムに用いられるストレージとして，分散ストレージが使われることが多い．分散処理システムで扱うデータ量は巨大になることが多く，1 台の計算機上でデータを保持することが難しいため，分散ストレージを利用することで保存可能なデータ総量がスケールアウト可能なためである．

分散ストレージに対して分散処理システムからのアクセスを実現させる場合には，専用のライブラリを用いるか NFS(Network File System) の様にネットワーク経由で分散処理システムの外部に存在する分散ストレージを計算ノード上にマウントする．

分散処理システムで利用する分散ストレージは，処理性能を発揮するために可能な限り高速にデータへとアクセス可能な設計を実現することが望ましい．そのためには，処理対象のデータが分散ストレージの何処に存在するかをソフトウェア開発者が強く意識し，データアクセスの高速化を考慮したソフトウェア開発を行わなければならない．

Hadoop システムのストレージである HDFS(Hadoop Distributed File System) を用いる YARN(Yet Another Resource Navigator) の様に，ミドルウェアがデータが存在するノードでの処理を積極的にスケジューリングし実行する機構も存在するが，その様な機構を使わない限りソフトウェア開発者はデータの場所を意識した処理記述を行う必要がありアクセス透過性の実現とアクセス性能を両立させることは難しい．ソフトウェア開発の効率を考えた場合には，分散ストレージへのアクセス透過性を実現することで開発効率が上がると推測される．

3. 提案手法

本研究では，

- 1 台の計算機環境と同等の処理を実行可能なスケールアウトする分散処理環境
- 1 台の計算機環境と同等のストレージアクセスを分散ストレージで行うアクセス抽象化
- 1 台の計算機環境と同等の処理を分散処理環境で実行可能なスケジューラ

を実現する．システムの内部状態を隠蔽しつつ処理を分散実行する基盤を構築することでソフトウェア開発者の，「リソース管理や処理実行スケジューリングを意識することにより発生する開発コストの問題」と「分散ストレージ上のデータに対するアクセス効率を意識した実装による開発コストに対する問題」の解決を目指し，「1 台の計算機で行う処理を，透過的にスケールアウト可能な分散処理システムで実行する基盤」の実現を考えた．これにより分散処理システムの内部状態や分散ストレージを意識しながら行うソフトウェア開発にかかる開発コストは抑えられる．

上記問題を解決するアーキテクチャを「ソフトウェア定義計算機」と定め，分散処理システムをあたかも 1 台の計算機と同等に利用可能なコンセプト実装を行った．

3.1 ソフトウェア定義計算機の構成

ソフトウェア定義計算機の構成を図 1 に示す．ソフトウェア定義計算機はソフトウェア開発者が開発したデータ解析ソフトウェアを動かす基盤として動作する．ソフトウェア定義計算機の分散ストレージには外部機器から収集された解析対象のデータが蓄積される．ソフトウェア開発

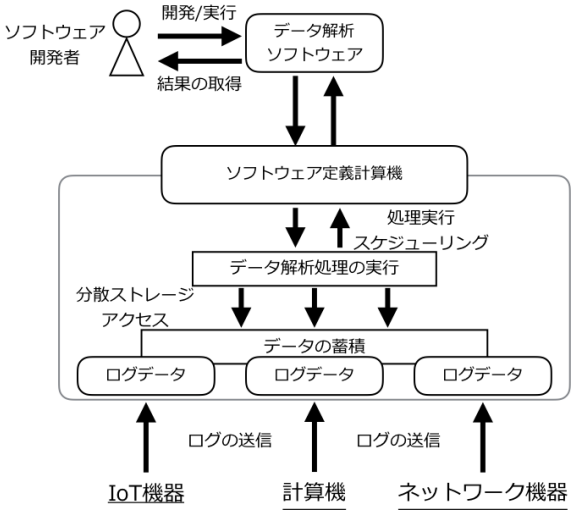

図 1　ソフトウェア定義計算機の構成

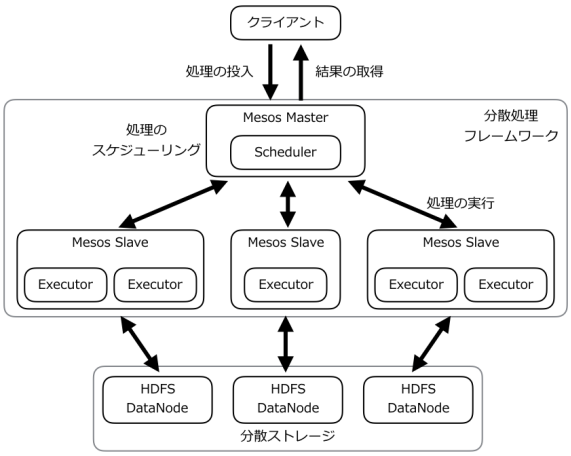

図 2　システムイメージ

者が開発したデータ解析ソフトウェアはソフトウェア定義計算機上で自動的にスケジューリングされ，プログラムから解析対象データに対して透過的にアクセスを行い解析を実行し結果を返す．

処理ターゲットは，大量に集められるネットワークに接続された機器から送付されるテキストデータ，ログデータを対象とし，データの抽出や集計を行うことを想定する．

3.2　システム構成

図 2 にシステムの詳細を示す．ソフトウェア定義計算機はシステム外部のクライアントから処理を投入することで動作を開始する．クライアントから投入された処理はスケジューラ (Scheduler) に渡され，スケジューラが立てる実行計画に組み込まれる．次にスケジューラから渡された処理を実行するために必要な計算資源管理を分散処理フレームワーク (Mesos) が行い，計算ノードへ処理を分配する．処理を渡された計算ノード (Executor) は，分散処理フレームワークにより確保された計算資源内で処理を実行し結果をスケジューラへと返す．計算ノードでは処理対象であるデータが保存される分散ストレージに対し，実行される処理から透過的にアクセスし計算を行う．処理結果は，計算ターゲットと同様に分散処理システムへと保存することが可能である．

3.2.1　分散処理フレームワーク: Mesos

本提案の基盤として分散処理フレームワークである Apache Mesos [6] をスケールアウトする分散処理システムの実行基盤として採用した．Mesos は DCOS(Data Center OS) と呼ばれるソフトウェアで，データセンターの計算資源を一元管理するためのミドルウェアとして開発された．Mesos はデータセンターに存在する Layer 2 ネットワークで接続された計算機群の計算機資源 (CPU コア数，メモリ量，ストレージ容量) を統合管理し，Mesos 上で動作するフレームワークが要求する資源量に応じて，計算資源を切り出しフレームワークへと提供して処理を実行する．Mesos 上で動作するフレームワークとして，Hadoop, Spark, MPI, Cassandra, Docker などのミドルウェアが存在する．これらミドルウェアは本来であれば個別に構築され利用されるが，Mesos により抽象化レイヤが提供され複数のミドルウェアが同時に動作する基盤が実現され，個別で構築する場合と比較して計算資源の有効活用が可能となる．

Mesos には処理実行を計画するスケジューラプログラム (Scheduler) と，実際に投入された処理を実行する実行プログラム (Executor) の大きく二つのプログラムがフレームワークを実現する上で必要となる．Scheduler と Executor の二つを Mesos 上で動作させることで，Mesos の計算資源管理と処理実行機構を利用することができる．

3.2.2　Scheduler と Executor

本提案では，Mesos 上で動作する Scheduler として Go 言語で記述した独自ソフトウェアと計算ノードで Scheduer から渡される処理を実行する Executor を実装した．また Scheduler へと行いたい処理を登録するプログラムとして，Go 言語で記述したクライアントを実装した．このクライアントは，処理を通常のソフトウェア開発者環境のシェルと同等のコマンドライン，パイプ，リダイレクト処理として受け取り，入力された一連のプログラム群を Scheduler へと渡すように動作する．

クライアントから登録された処理は，Scheduler 内の処理キューへと登録され，Mesos の資源管理機構により計算資源が提供され Mesos に管理される Executor 上で実行される．

3.2.3　分散ストレージ: HDFS

本提案のストレージとして，分散ストレージとして動作する HDFS を採用した．HDFS へのアクセスは通常 Hadoop から提供される専用コマンドラインや，WebHDFS など Web 経由で利用可能な REST API や，HDFS 専用 API を用いたライブラリを利用して行う．

表 1　ノード追加時のリソース総量の増加

Slave ノード台数	core 数合計	メモリ量合計
1 台	24	46GB
5 台	120	230GB
10 台	240	460GB

本提案では Mesos のフレームワークを用いることから，Executor から HDFS 内部を直接参照可能なように，HDFS へのアクセスを Executor が実行されるノードからローカルストレージへのアクセスと同等に見せるための hadoop-hdfs-fuse を利用した FUSE(Filesystem in Userspace) を用いて実現した．

4. 結果

本研究の達成項目として以下を挙げる．

- NICT StarBED [7] を用いた Mesos の動作検証と分散処理環境としてのスケールアウト性能や耐障害性の確認
- HDFS による分散ストレージの構築と Mesos の計算ノードから HDFS への透過的アクセスを FUSE を用いて実証
- Scheduler と Executor のプロトタイプ実装と，投入された処理が Mesos の計算ノードへと配布され HDFS へと透過的にアクセス可能なことを確認

4.1 スケールアウト検証

Mesos で動作する計算機は大きくクラスタ全体のリソース管理を司る Master ノードと計算ノードとして動作する Slave ノードの 2 つに別れる．Master ノードは Zookeeper [8] を用いた 3 台のノードとして冗長化構成が取られ，障害が発生したとしても Master ノードで提供される資源管理機能や Slave ノード管理機能が即座に他の正常なノードへと切り替わり動作する．Slave ノードは計算ノードとして動作し，Master ノードに対して現在どの程度の CPU, メモリ，ディスクなどの計算資源が利用されているかを定期的に通知する．Slave ノードが Master ノードから処理を割り当てられた場合には，必要な計算資源をロックして処理を実行し，その処理が終わったタイミングで Master ノードへと処理終了 (場合によっては異常終了) を通知して，ロックしていた計算資源を解放する．

新たな Slave ノードが Master ノードが管理するクラスタへ追加されると，Master ノードは追加された Slave ノードを自動的に検知し計算機リソースの 1 台として管理を開始し計算資源の一部として利用する．

表 1 は，NICT 北陸 StarBED 技術センターで行った Mesos のスケールアウト実験結果で，Slave ノードを複数台動作させた時の CPU core 数，メモリ数の合計推移を表す．

Slave ノードを追加することにより，Master ノードで管理する総計算資源量が増加することがわかる．また Slave ノードが故障した場合には，自動的に Master ノードによって故障ノードが切り離され計算資源の総量は減少する．

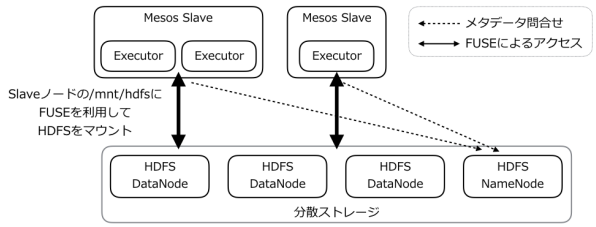

図 3　Slave ノードからの FUSE を使った HDFS へのアクセス

Slave ノードのスケールアウトは Mesos フレームワークで容易に行うことができるが，ストレージのスケールアウトは分散ストレージである HDFS を利用して行った．HDFS は対障害性を高めるために，データのレプリカを 3 つ作成して複数ノードで保持し冗長化を行う (レプリケーション数は設定可能であるが，本提案では 3 つとした)．レプリカの数が 3 ということはシステム全体としてコピーが 3 つ作られることとなり，ストレージの総量としてはノード全体が保持するストレージ量の 1/3 ほどになるが，HDFS のノードを増やすことによりデータ総量を増加させることでスケールアウト可能である．

4.2 FUSE を用いた HDFS へのアクセス

Mesos の Slave ノードである計算ノードから HDFS へのアクセスを実現するには幾つか方法がある．計算ノードで実行されるプログラムから直に HDFS のデータを保持するノード (DataNode) を利用するライブラリを用いる方法と，FUSE を使ってユーザ空間から透過的に DataNode へとアクセスする手法である．

前者は，プログラムを記述するソフトウェア開発者が意識的に HDFS のメタデータを保持する NameNode の場所を把握し，NameNode からどこにアクセスを行いたいデータが存在するか情報を引き出してから DataNode へアクセスすることになり，HDFS を強く意識してデータの処理を記述する必要がある．記述するプログラミング言語により利用するライブラリが異なることから，ソフトウェア開発者は自身が記述するプログラミング言語に沿ったライブラリを利用して開発を行う必要がある．

本提案では，図 3 に示すように FUSE を用いた HDFS へのアクセスを利用することで，ソフトウェア開発者が開発するプログラムから NameNode に対するメタデータ問い合わせを隠蔽した．これによりソフトウェア開発者はローカルファイルアクセスと同様に HDFS が利用可能となり，プログラム言語毎の HDFS ライブラリの差異を気にせずに処理を記述することが可能となる．

本システムは以下のコマンドにより各 Mesos Slave ノー

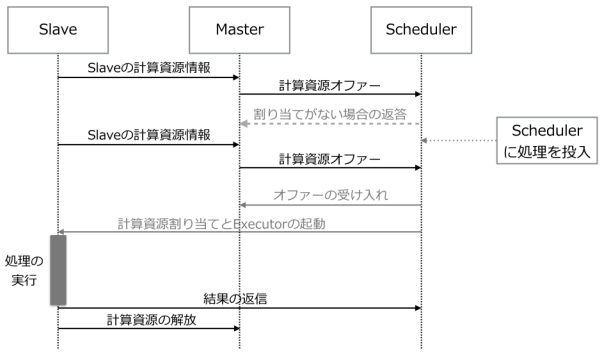

図 4　Mesos 内部での資源割り当て

ドから HDFS をマウントしている．

```
# /usr/bin/hadoop-fuse-dfs /mnt/hdfs -o rw,\
server=hdfs://NameNode ip address,port=8020
```

FUSE を利用することでファイルの追記が不可能になるが，本システムでは FUSE を用いたアクセスにおけるファイルの追記に関しては範囲外とした．

4.3　Scheduler の動作順序と資源割り当て

図 4 が Mesos の処理割り当て順序となる．ソフトウェア開発者がクライアントから Scheduler へ処理を投入すると，投入された処理は Scheduler 内でキューイングされ Mesos の Master ノードによって資源割り当てが行われる．Mesos の Master ノードは Slave ノードから常に計算資源の利用状況を提供されており，Scheduler から処理要求がきた場合には資源割り当て可能な Slave ノードを選択して処理を割りあてる．資源を割り当てられた処理タスクは Mesos の Slave ノード内で，処理実行機構である Executor により実行される．

例として以下の処理が Scheduler へ投入された場合の動作を解説する．

```
$ ./sdc-client /bin/cat /file/1.txt \
/file/2.txt /file/3.txt
```

クライアントである sdc-client コマンドの引数は以下のコマンドに分解される．

```
/bin/cat /file/1.txt
/bin/cat /file/2.txt
/bin/cat /file/3.txt
```

3 つのコマンドに分解された処理は, Scheduler 内部でキューイングされ Master ノードによって資源が割り当てが行われる．Master ノードは，計算資源に余裕がある Slave ノード群に対し 3 つの処理を割り当て結果を取得する．Master ノードによる処理の割り当てはその時に余裕がある Slave ノードの計算資源に依存するため，各処理がどの Slave ノードで実行されるかの保証はなく同一ノードで実行される可能性もある．

4.4　Slave ノードから HDFS への透過的アクセス

Mesos の Slave ノードで実行される Executor から分散ストレージである HDFS へとアクセスが必要な場合は，FUSE を使うことにより Slave ノードのローカルファイルシステムにアクセスする方法と同様に HDFS のディレクトリにアクセス可能である．Executor で処理した結果を保存したい場合にも FUSE を利用して HDFS へと書き込むことができる．

HDFS を透過的に利用するコマンド例は以下となる．

```
$ ./sdc-client /bin/cat /hdfs/1.txt \
/hdfs/2.txt /hdfs/3.txt > /hdfs/result.txt
```

この例では，/hdfs ディレクトリが FUSE を用いて Slave ノード上にマウントした HDFS ディレクトリであり，そのディレクトリに対するファイルの読み込みと，処理結果をまとめてファイルへと書き込むことができる．このようにソフトウェア開発者は HDFS を意識せずに通常のファイルアクセスと同等な処理を投入することができる．

5.　まとめと今後の課題

5.1　まとめ

ソフトウェア開発者が分散処理システムを意識せずに利用可能な基盤としてソフトウェア定義計算機のコンセプト実装を，Mesos と HDFS を利用し，オリジナルのフレームワークとして Scheduler, Executor を実装することにより実現した．ソフトウェア定義計算機の実装を行った結果問題となった項目として以下が挙げられる．

- パイプラインで接続した shell script をスケジューラに投入した際にコマンド実行順序が保証されない問題
- 透過的ストレージアクセスを実現するために採用した FUSE のストレージアクセス遅延

それぞれにの問題ついて解説を行う．

5.2　実行コマンドの順序保証

クライアントから投入された処理がどのように実行されるかは，実行されるシステムやプログラムに依存する．投入された処理が処理環境によっては，ソフトウェア開発者の意図した順序に処理が進まない可能性がある．また並列実行を行いたい場合にも，投入された処理よっては並列性は保証されずに，ソフトウェア開発者が期待する並列度で動作しない可能性もある．順序が保障されない場合に，並列度を高めようとしても並列化ができる部分が定まらないため，結局は逐次実行され期待した並列化は行われない．

以下のコマンドを例示する．

```
$ grep xyz *.txt
```

「*.txt」の展開のされ方は，実行するシェル環境やシステムによって大きく変化したり，実行するたびに処理順序が変わる可能性がある．処理の順序を保証しかつ並列実行を保障するためには，処理の分解と計算する対象のデータ量の推測，さらには分解した処理自身が他のどの処理に依存するかというグラフを解く必要がある．

5.3 FUSE のアクセス遅延

ソフトウェア開発者のローカル環境と FUSE を用いた分散ストレージで行う，テキストファイルから特定のキーワード検索と集計処理を比較した場合に処理時間には大きな差がある．以下は，ファイル数 291，データ総量 2.7GB，データ総行数 286 万行のデータに対して grep コマンドを実行した結果である．

```
$ time grep mod_ssl *.dat | wc -l
1009960

real    0m2.253s
user    0m1.507s
sys 0m0.974s
```

ローカル環境で行った場合には処理は 2.2 秒程度で終了している．

```
$ time grep mod_ssl /mnt/hdfs/testdata/*.dat
\ | wc -l
1009960

real    0m35.826s
user    0m1.748s
sys 0m1.757s
```

HDFS に対して FUSE でアクセスした時に同様の処理を行った場合にかかる時間は 35 秒ほどで，ローカルファイルシステムの 17 倍ほど時間がかかる．FUSE を用いた場合，ローカルアクセスと比較して分散ストレージへのアクセスにかかるオーバヘッドが大きい．分散処理システムでのストレージアクセスの遅延は全体の処理速度に大きく影響するため，FUSE を利用してストレージアクセスへの透過性を実現した場合，処理速度が大きく低下することが懸念される．

5.4 今後の課題

ソフトウェア定義計算機を用いて高パフォーマンスなデータ処理を実現するためには，分散処理に特化した分散処理実行スケジューラの実現が必要となる．スケジューラが処理する実行順序を分散処理システムへと最適化するために，DAG(Directed Acyclic Graph) アルゴリズムがシステムに組み込めるか検討を行った．また，FUSE を用いたストレージアクセスの遅延を解消するために，HDFS プロトコルを解析しデータ保存場所と処理実行ノードの関係を強く意識したストレージノード上でのデータ処理に関するアーキテクチャを再考した．

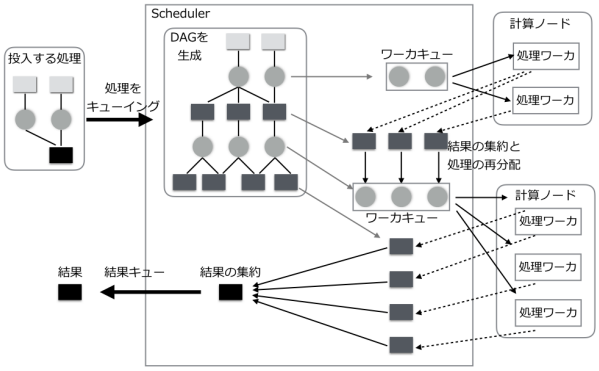

図 5　DAG を用いたスケジューラ

5.5 スケジューラへの DAG の組み込み

DAG(Directed acyclic graph) を用いた分散処理実行アルゴリズムと，処理対象データの分割と処理の粒度を考慮したプロセスの分割を行う改良を思考した．DAG を用いることで実行する処理を，向きがあり循環の無いグラフとして並列性を考慮したタスクに分解することができる．

図 5 で示すようにスケジューラへと投入される処理はスケジューラ内部で，DAG アルゴリズムにより小さな処理へと分解され，並列化が行える部分に関しては計算ノードへと分割して渡され結果の集約を行う．処理の分解と集約が多段に発生する場合には，分解された処理は計算ノードへと渡され結果をスケジューラで集約するという動作を繰り返す．最終結果をスケジューラがまとめあげ，処理結果をクライアントへと返す．スケジューラの処理の分解粒度に関しては処理対象となるデータサイズに依存するため，計算ノードのメモリに乗り切らないような巨大なデータを処理する場合には，さらに処理を分解してメモリに収まるサイズで処理を行う必要がある．

5.6 処理速度を高めるための HDFS の利用方法

FUSE を利用した結果，HDFS への透過的アクセスは実現したがアクセス速度が低下した．現在どの DataNode がどのデータを保持しているかは，NameNode へと API 問い合わせを行うことで確認できる．計算ノードが対象となるデータへの処理速度を高速化する方法は，データを保持するノード自身で処理を行うことである．つまり，Mesos

の計算ノードである Slave ノードと，HDFS の DataNode を同じ計算機上で共存させ，Mesos のスケジューラが対象データを保持するノードへ処理を渡すことで，近い場所で処理を行い結果を返すことができ処理速度が向上する．

5.7 開発環境と分散処理システム環境の透過性実現の提案

5.7.1 開発環境と分散処理システム環境

分散処理システム上で開発を行う場合は問題ないが，ソフトウェア開発者は，最初から実際の分散処理システム上で開発を行わない可能性が高い．ソフトウェア開発者は仮想マシンや Docker などのコンテナ技術を用いて開発を行い，開発したソフトウェアを本番環境である分散処理システムへと移し実行する．しかしながら，実環境である分散処理システムと同等の環境をソフトウェア開発者が利用する開発環境に構築するには，時間的コストや物理的な計算機資源の制約，環境の差異など幾つかの問題が発生する．ソフトウェア開発者が普段利用する開発環境で，分散処理システムと同等の処理が記述可能なライブラリ群を利用することができればこれらの問題は解決する．

そこで，実環境と同等の DAG を組み込んだスケジューラと HDFS アクセスを透過的に可能にするライブラリを開発環境で用いることでこれらを実現する．DAG スケジューラとして，Python の DASK ライブラリ [9] を用いることで，ユーザの環境において処理したいデータに対し自動的にプロセスの分解を行い結果を返すことができる．DASK ライブラリは Mesos に対応していないため，分解された処理を Mesos の Slave ノードで実行できるよう改良が必要となる．

5.7.2 HDFS の透過ライブラリ

HDFS アクセスの隠蔽に関しては，TFS(Transparent File System)[10] を用いることで解決できる．このライブラリは，Python の HDFS3 ライブラリで提供される API を内包したライブラリであり，HDFS で行うディレクトリ/ファイル操作を，ソフトウェア開発者の開発環境で行うディレクトリ/ファイル操作に置き換えることができる．ソフトウェア開発者は HDFS を利用する場合に設定ファイルへ HDFS の NameNode 情報を記述するだけで，ディレクトリ/ファイル操作を開発環境から HDFS 環境へと切り替えることができる．ディレクトリ/ファイル操作を記述するプログラムに変化は起こらずコード記述レベルの透過性が実現できる．TFS は本システム用にソフトウェア開発者の開発環境と分散処理システム環境を橋渡しするために実装したコンセプトライブラリである．

6. 課題を反映した新しいアーキテクチャ

5.4 章で述べたように，

- スケジューラへの DAG を用いたアルゴリズムの導入
- FUSE を用いない HDFS の透過的利用の仕組みの導入

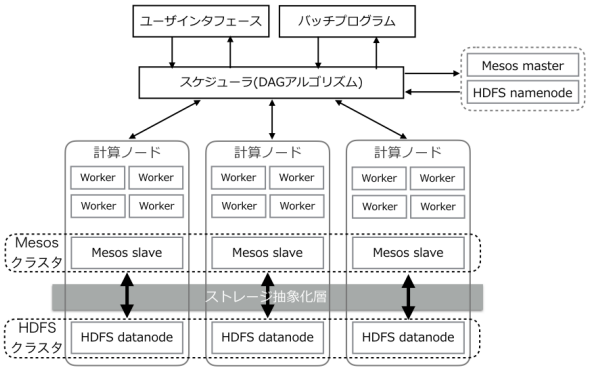

図 6 新しい分散処理アーキテクチャ

- ソフトウェア開発者自身の開発環境と同等の処理を動作させるライブラリの実現

の 3 点を目的とし，本提案で問題となった課題を解決する新しいアーキテクチャの設計を行った．新アーキテクチャとして 図 6 を示す．新しいアーキテクチャでは，Mesos の Slave ノードと HDFS の DataNode が同じ計算機上に存在する．スケジューラはデータの距離を考慮し，データが存在する一番近くのノードへと処理を投入する．さらに HDFS のレプリカが異なる DataNode へと 3 つ作成される特性を生かし，特定の計算ノードへと投入される処理が集中しないように，レプリカの場所と計算ノード上で動作する処理総数をスケジューラが監視しつつ処理をシステム全体へとバランスさせ，計算ノードの処理負荷が偏らないようにする．新しいアーキテクチャの評価に関しては今後の課題とする．

7. 謝辞

本研究にあたり，情報通信研究機構 北陸 StarBED 技術センターの計算機群やネットワーク設備を利用させていただいたことに，深く感謝いたします．また設備の使い方や研究のアドバイスなど技術員，研究員の方々にも多大なお力添えをいただいたことに深く感謝いたします．

参考文献

[1] Apache Hadoop (online), http://hadoop.apache.org/, 2016-05.
[2] Apache Spark (online), http://spark.apache.org/, 2016-05.
[3] Apache Hive (online), https://hive.apache.org/, 2016-05.
[4] Apache Impala (online), http://impala.io/, 2016-05.
[5] Dean, J., Ghemawat, S., 2008. Mapreduce: simplified data processing on large clusters. Communications of the ACM 51, 107-113.
[6] Hindman, B., Konwinski, A., Zaharia, M. Ghodsi, A, Joseph, A.D., Katz, R.H., Shenker, S., Stoica, I., Mesos: a platform for fine-grained resource sharing in the data center. In: NSDI, vol.11, pp.22-22(2011)
[7] 情報通信研究機構 北陸 StarBED 技術センター, http://starbed.nict.go.jp/, 2016-05.
[8] Apache Zookeeper online),

https://zookeeper.apache.org/, 2016-05.
[9] Dask is a flexible parallel computing library for analytics (online), http://dask.pydata.org/en/latest/, 2016-05.
[10] TFS(Transparent File System) (online), https://github.com/hirolovesbeer/tfs, 2016-05.