Asakusa Framework-RDBMS 連携方法

2012/10/12

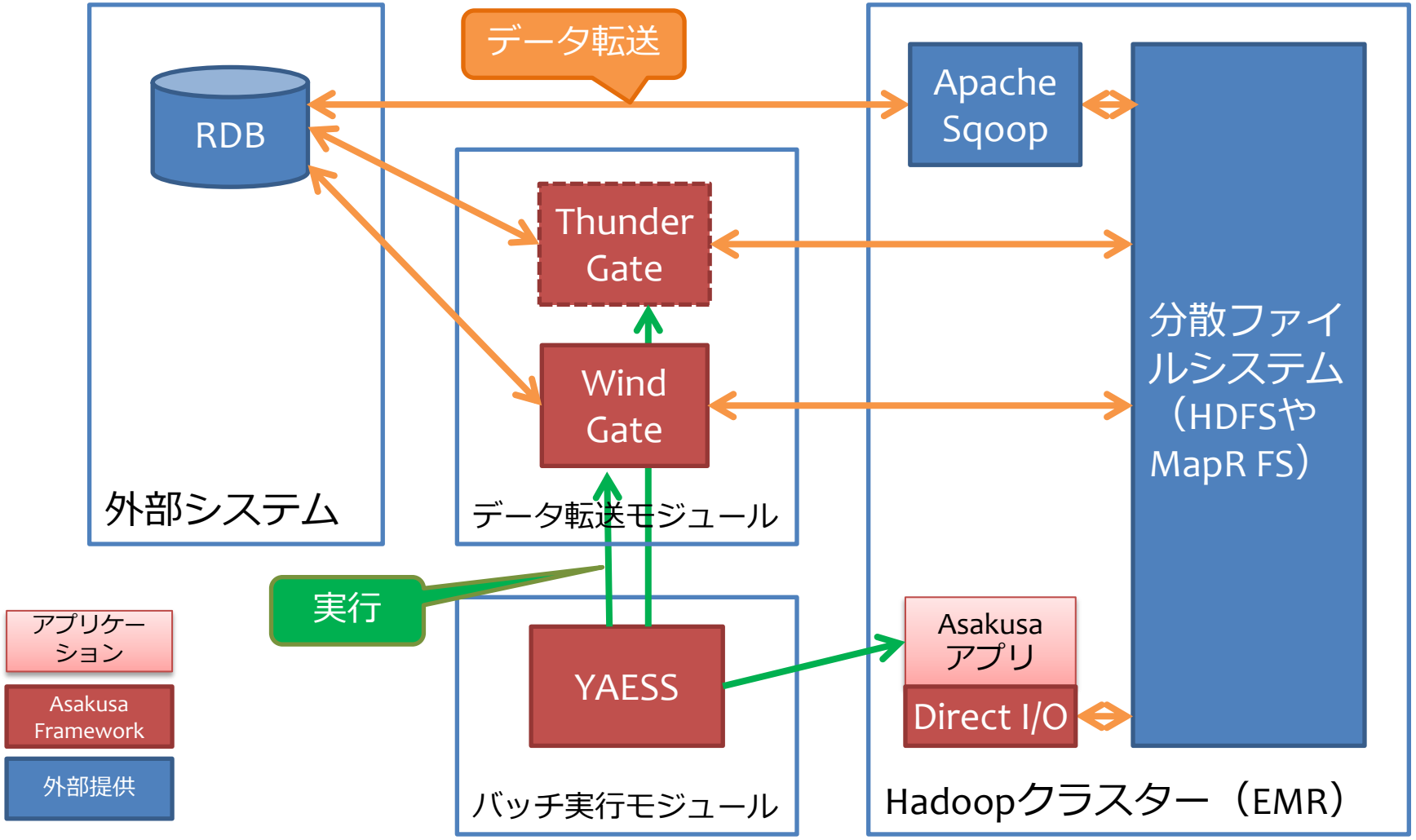
目次

1. 概要
2. 連携パターン
3. ベンチマーク測定環境
4. 測定結果
5. まとめ

概要

- Asakusa Frameworkを使用したアプリケーションとRDBMSの連携方法を例示する。
  - Asakusa FrameworkにはRDBMSと連携する為のコンポーネント（ThunderGateやWindGate）が在るが、RDBMSとの連携方法はそれだけとは限らない。
    当資料でその例を示す。
  - 例示した内のいくつかのパターンについて、実際にAWS（EMR）で環境を構築して実行時間を測定した。
    システム構成を決定する要因は実行速度だけではないが、参考までに掲示する。

当資料の前提となるコンポーネント・プロダクト



コンポーネント

- WindGate（Asakusa Framework0.4）
  - RDBやローカルファイルにアクセスする為のコンポーネント
    - JDBC接続でRDBMSのテーブルを読み書きできる。
    - ローカルのCSVファイルを読み書きできる。
- ThunderGate（Asakusa Framework0.4）
  - MySQLにアクセスする為のコンポーネント
    - トランザクション機能等を持つが、各テーブルに専用カラムを設けたり専用テーブルを作ったりする必要がある。

- Direct I/O（Asakusa Framework0.4）
  - HDFSやAmazon S3上のファイルを（WindGate等を経由せずに直接）読み書きする機能
    - これは機能名であって、「Direct I/O」というコンポーネントがあるわけではない。
- YAESS（Asakusa Framework0.4）
  - Asakusa Frameworkの各コンポーネントやAsakusaアプリケーションを実行するコンポーネント

- Asakusaアプリケーション
  - Asakusa Frameworkを使用して作成されたアプリケーション

プロダクト

- Apache Hadoop
  - 分散処理基盤
    - Asakusa Framework0.4では、Asakusa DSLで記述されたアプリケーションをHadoopのMapReduceに変換して実行する。
    - Apache Hadoopのディストリビューションやサービスとして CDH（Cloudera's Distribution including Apache Hadoop）や MapR・Amazon EMR等がある。
- Apache Sqoop
  - RDBMSとHDFSを連携するツール
    - RDBのテーブルを読み込んでHDFS上にファイルを作る、またHDFS上のファイルを読み込んでテーブルに書き込む。
    - HadoopのMapタスクで分散して処理する。
    - 扱えるファイルはCSVファイル・SequenceFile等。

プロダクト（サービス）

- MapR
  - MapR Technologies社が開発したHadoopディストリビューション
    - HDFSの代わりにMapR FSという独自の分散ファイルシステムを使用する。また、これをNFSマウントしてローカルファイルシステムの様に扱うことが出来る。
      - 例えば、NFSマウントした領域にUNIXコマンドでファイルを作ると、MapR FS上のファイルとしてMapReduceプログラム（Asakusaアプリケーション）から読み込むことが出来る。
    - Amazon EMRのサービスでも、稼働させるHadoopの種類としてMapRを選択することが出来る。

サービス

- Amazon Web Services（AWS）
  - 仮想マシンを構築・使用できるサービス
- Amazon Elastic Compute Cloud（EC2）
  - 仮想マシン
- Amazon Elastic MapReduce（EMR）
  - Hadoopクラスターを構築・使用できるサービス
    - Apache Hadoopカスタマイズ版とMapR版の二種類が使用できる。
    - 使用する度にEMRを起動する使い方だと、HDFS（MapR FS）上に永続的なデータ保持をすることは出来ない。永続的なデータ保持の為にS3を利用することが出来る。
- Amazon Simple Storage Service（S3）
  - ファイルを格納するサービス

考えられるAsakusaアプリケーションとRDBMSとの連携方式

連携パターン

AsakusaアプリケーションとRDBMSの連携パターン

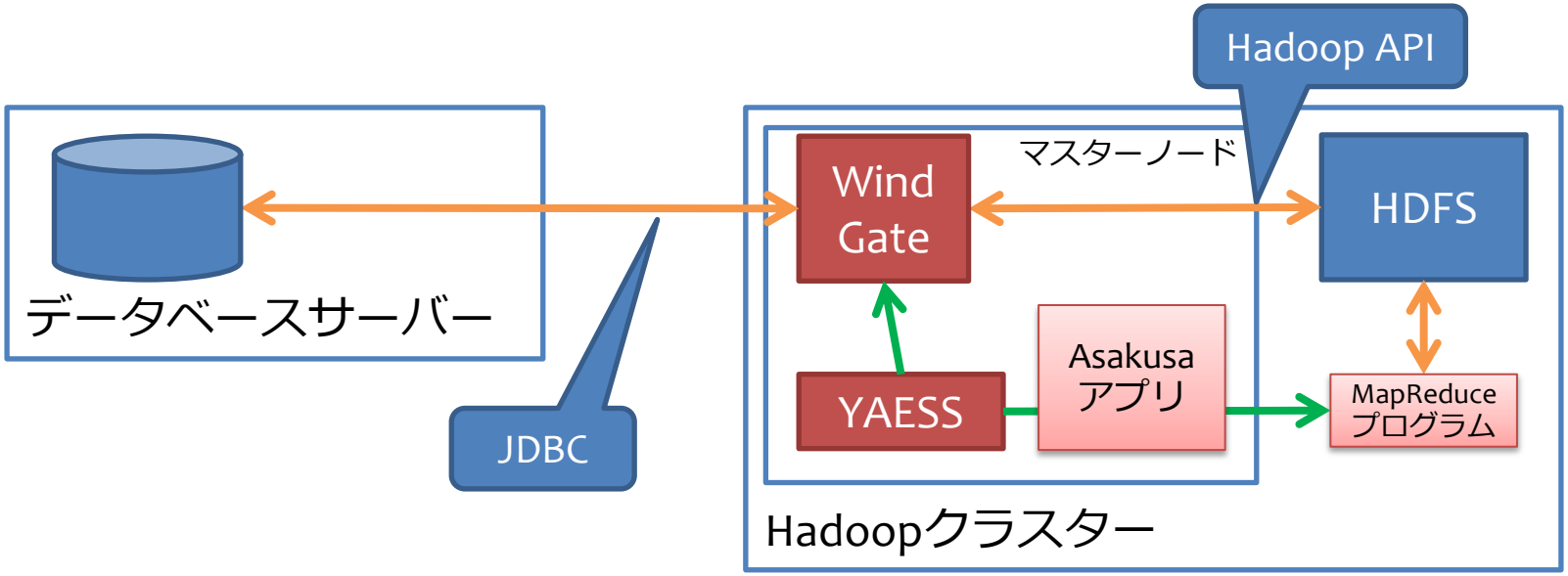
- 以下のような連携パターンが考えられる。
  1. 直接DBアクセス
     - WindGateでJDBCにより直接DBアクセス
       - Asakusa Framework以外のプロダクトを必要としない。
  2. RDBMS連携ツール経由
     - RDBMS連携ツール（Sqoop等）でHDFS上にファイルを作ってDirect I/Oで読み書き
       - Asakusa Framework以外のツールをインストール・設定する必要があるが、RDBMS連携に特化したツールであれば、機能や実行性能が良いことが期待できる。

3. RDBダンプツールやRDBロードツール経由

- ローカルのCSVファイルをWindGateで読み書き
- CSVファイルをHDFSに転送してDirect I/Oで読み書き
- CSVファイルをAmazon S3に転送してDirect I/Oで読み書き
- CSVファイルをMapR FSに転送してDirect I/Oで読み書き
  - 一般に、RDBMSが提供しているダンプツールやロードツールを使って一括処理する方が、SQLによる入出力より性能が良い。
    - » ダンプツール：テーブルから読み込んでローカルファイルシステム上にCSVファイルを生成する想定
    - » ロードツール：ローカルファイルシステム上のCSVファイルを読み込んでテーブルに書き込む想定（いわゆるバルクロード）
  - Asakusaアプリケーションからファイルを扱うには、WindGateでローカルファイルを読み書きする方式や、Direct I/OでHDFS・MapR FSやS3上のファイルを読み書きする方式が考えられる。

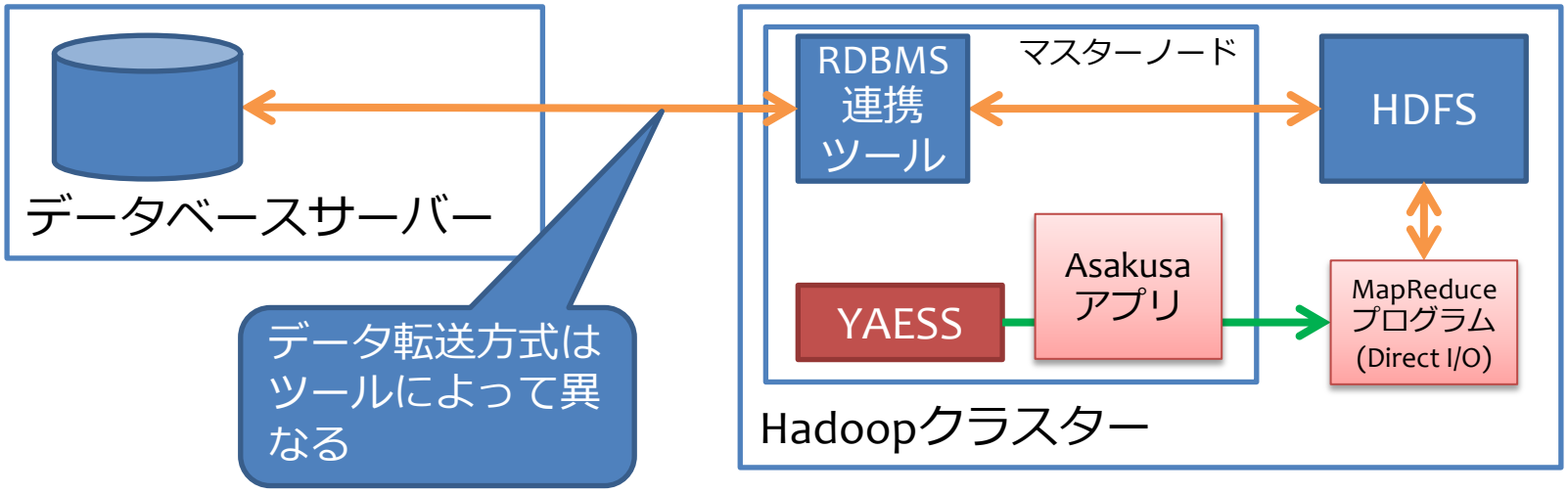
1-1. WindGate JDBC

- 概要
  - WindGateのJDBC接続機能でRDBMSにアクセスする。
- 構成の特徴
  - JDBC接続で直接RDBMSにアクセスする。
    - 標準的なJDBC・SQLだけ使用しているので、様々なRDBMSが使用可能。逆に、性能面で優れているRDBMS固有機能があっても使用できない。
  - DBサーバーにHadoopやAsakusa Frameworkをインストールする必要が無い。
  - WindGateが置かれているサーバーからDBサーバーへアクセスできる必要がある。



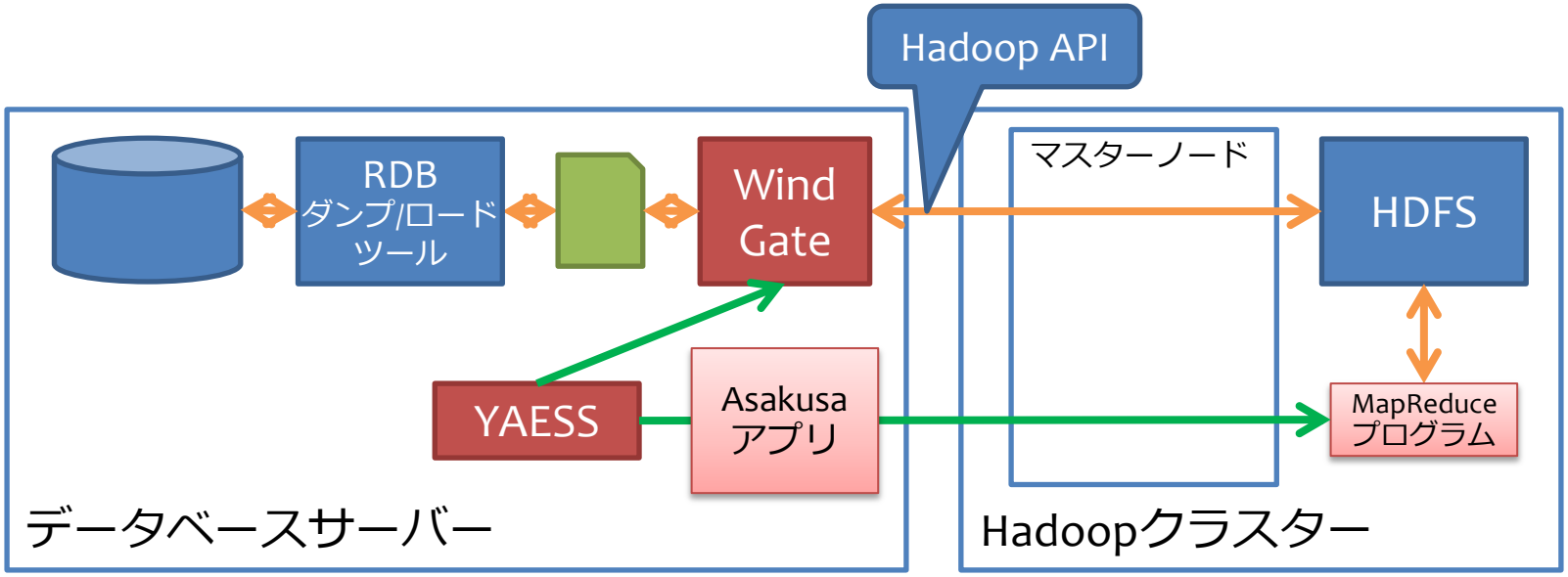
2-1. Direct I/O＋RDBMS連携ツール

- 概要
  - RDBMS連携ツール（Sqoop等）でHDFS上にファイルを作成し、AsakusaアプリケーションのDirect I/Oで読み込む。
  - AsakusaアプリケーションのDirect I/OでHDFS上にファイルを作成し、RDBMS連携ツールでDBに反映する。
- 構成の特徴
  - RDBMSとHDFSの連携（データ転送）に特化したツールを使用する。
    - データ連携専用のツールなので、性能面や機能面が良いことが期待できる。
  - Asakusa Frameworkの他にRDBMS連携ツールをインストール・設定する必要がある。
  - RDBMS連携ツールが稼働するサーバーからDBサーバーへアクセスできる必要がある。
    - 例えばSqoopは基本的にMapReduceでDBアクセスするので、全データノードからDBサーバーへアクセスできる必要がある。



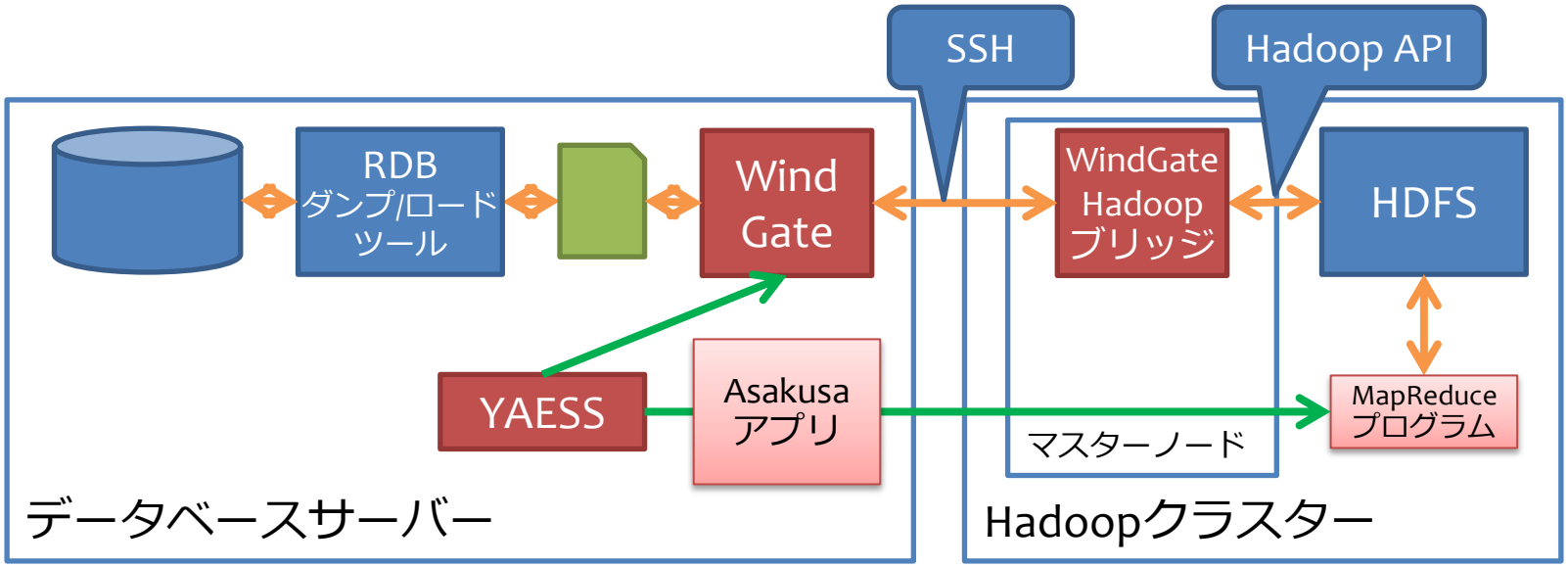
3-1-1. WindGate File＋RDBダンプ/ロードツール

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、WindGateのローカルファイルアクセス機能で読み込む。
  - WindGateのローカルファイル作成機能でDBサーバー上にCSVファイルを作成し、RDBロードツールでDBに反映する。
- 構成の特徴
  - WindGateをDBサーバーに配置し、DBサーバー上のCSVファイルをWindGateで読み書きする。
  - DBサーバーにAsakusa Framework（およびHadoop）をインストール・設定する必要がある。
  - DBサーバーからHadoopクラスターへアクセスできる必要がある。
  - ファイルを格納できるだけのディスク容量がDBサーバーに必要となる。



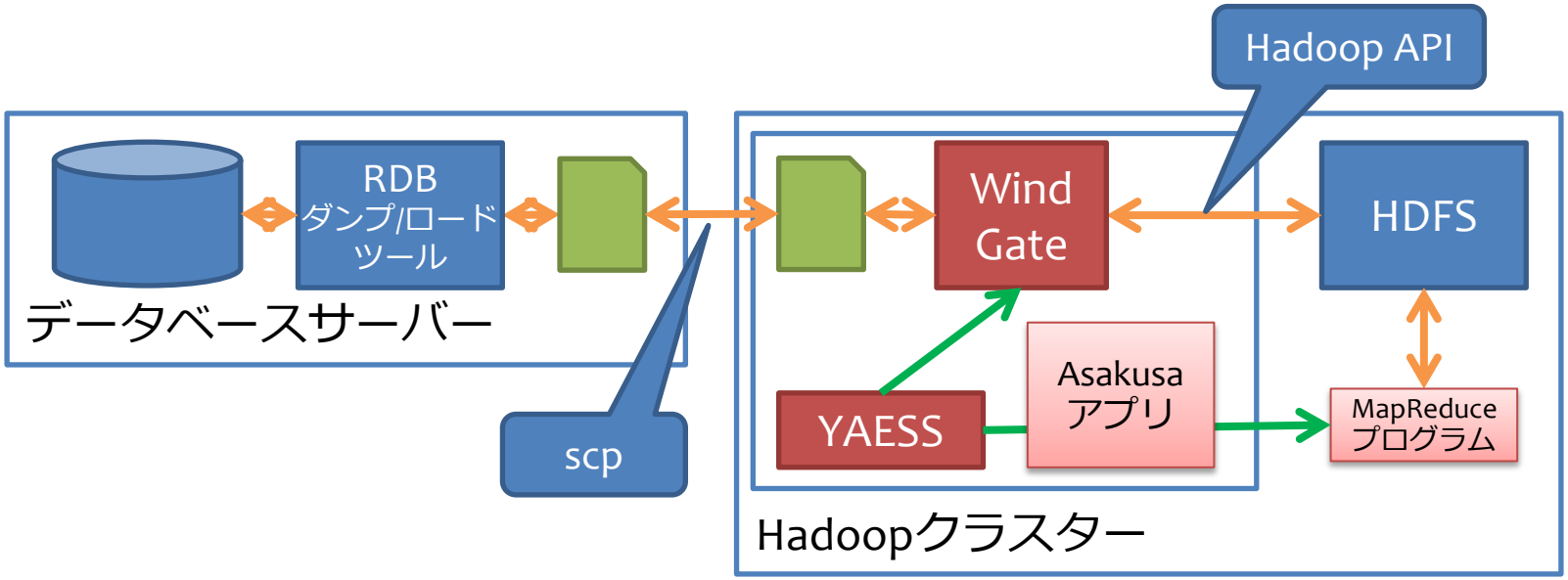
3-1-2. WindGate SSH＋RDBダンプ/ロードツール

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、WindGateで読み込む。
  - WindGateでDBサーバー上にCSVファイルを作成し、RDBロードツールでDBに反映する。
- 構成の特徴
  - WindGateをDBサーバーに配置し、WindGateのHadoopブリッジをHadoopクラスターに配置する。
    - WindGateユーザーガイドの『SSH経由でリモートのHadoopを利用する』
      http://asakusafw.s3.amazonaws.com/documents/0.4.0/release/ja/html/windgate/user-guide.html#sshhadoop
  - DBサーバーにAsakusa Framework（およびHadoop）をインストール・設定する必要がある。
  - HadoopクラスターにWindGateのHadoopブリッジをインストール・設定する必要がある。
  - DBサーバーからHadoopクラスターへSSH接続できる必要がある。
  - ファイルを格納できるだけのディスク容量がDBサーバーに必要となる。



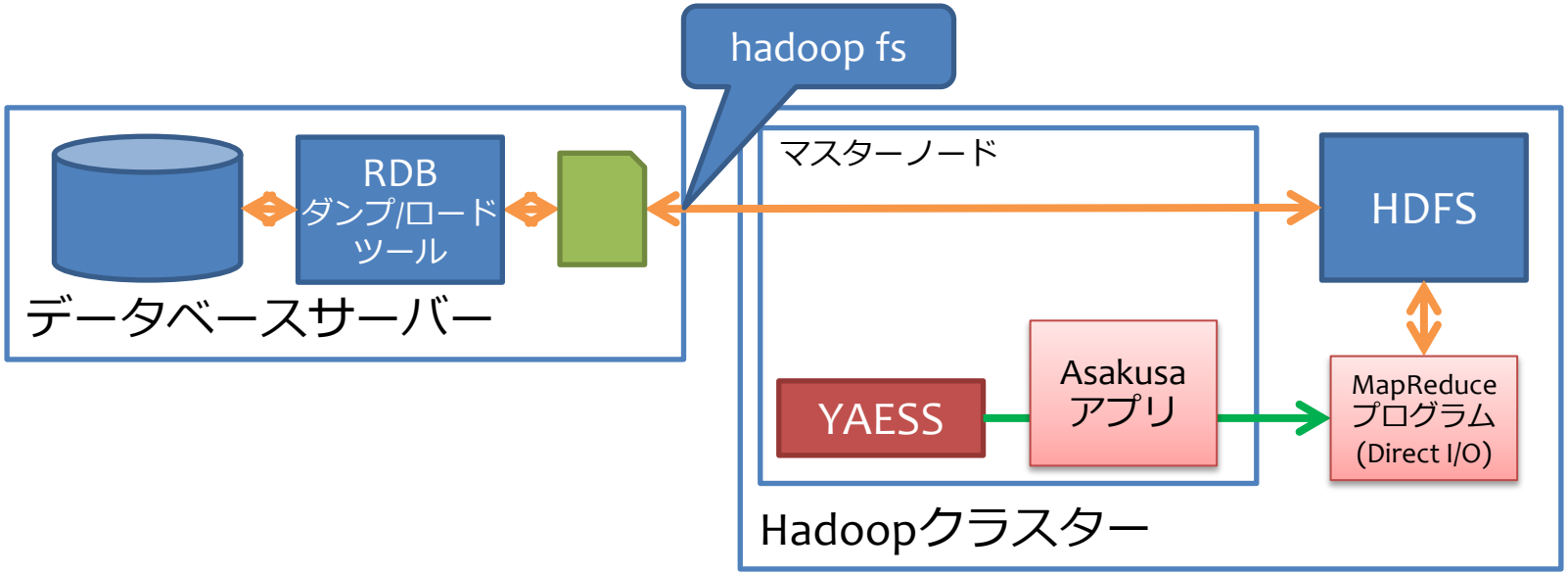
3-1-3. WindGate File＋RDBダンプ/ロードツール＋転送

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、scp等でHadoopマスターノードに転送してWindGateのローカルファイルアクセス機能で読み込む。
  - WindGateのローカルファイル作成機能でHadoopマスターノード上にCSVファイルを作成し、scp等でDBサーバーに転送してRDBロードツールでDBに反映する。
- 構成の特徴
  - WindGateをHadoopマスターノードに配置し、マスターノード上のCSVファイルをWindGateで読み書きする。
  - DBサーバーにHadoopやAsakusa Frameworkをインストールする必要が無い。
  - DBサーバーとHadoopクラスター間でデータ転送できる必要がある。（scpの場合はSSH接続できる必要がある）
  - ファイルを格納できるだけのディスク容量がDBサーバーおよびHadoopマスターノードに必要となる。



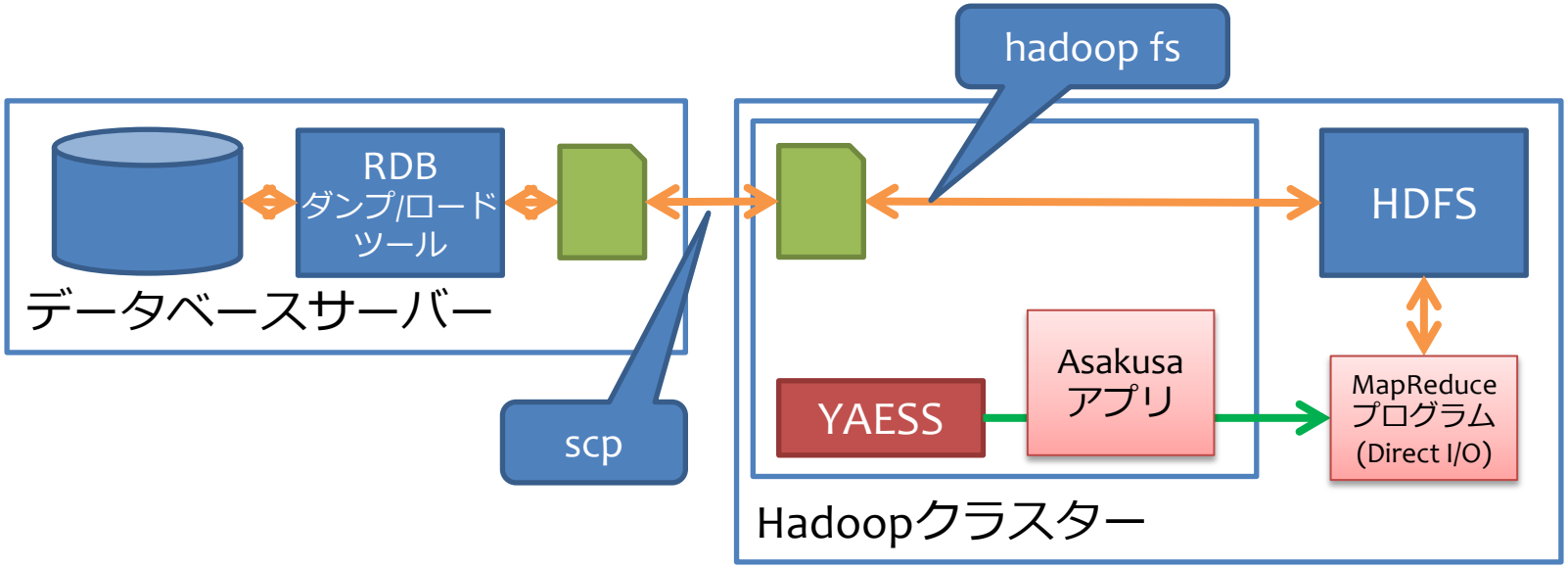
3-2-1. Direct I/O＋RDBダンプ/ロードツール＋HDFS

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、HDFSへ転送するツール（hadoop fs –putコマンド等）でHDFSに転送してAsakusaアプリケーションのDirect I/Oで読み込む。
  - AsakusaアプリケーションのDirect I/OでHDFS上にCSVファイルを作成し、HDFSから転送するツール（hadoop fs –getコマンド等）でDBサーバーに転送してRDBロードツールでDBに反映する。
- 構成の特徴
  - DBサーバーからHDFSへ直接アクセスする。
  - DBサーバーにHadoopをインストール・設定する必要がある。
  - DBサーバーからHadoopクラスターへアクセスできる必要がある。
  - ファイルを格納できるだけのディスク容量がDBサーバーに必要となる。



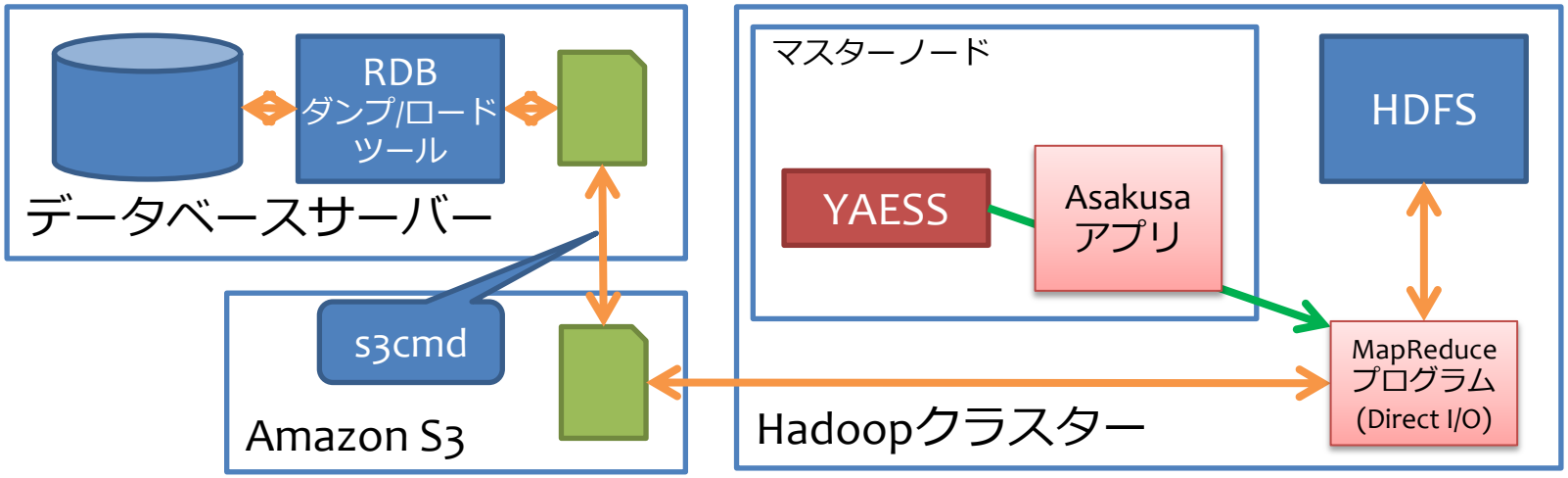
3-2-2. Direct I/O＋RDBダンプ/ロードツール＋転送＋HDFS

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、scp等でHadoopマスターノードに転送して、 HDFSへ転送するツール（ hadoop fs –putコマンド等）で転送してAsakusaアプリケーションのDirect I/Oで読み込む。
  - AsakusaアプリケーションのDirect I/OでHDFS上にCSVファイルを作成し、HDFSから転送するツール（hadoop fs –getコマンド等）で一旦ローカルファイルシステムに転送してから、scp等でDBサーバーに転送してRDBロードツールでDBに反映する。
- 構成の特徴
  - 3-1-3「 WindGate File＋RDBダンプ/ロードツール＋転送」と似ているが、WindGateでなくDirect I/Oを使う。
  - DBサーバーにHadoopやAsakusa Frameworkをインストールする必要が無い。
  - DBサーバーとHadoopクラスター間でデータ転送できる必要がある。（scpの場合はSSH接続できる必要がある）
  - ファイルを格納できるだけのディスク容量がDBサーバーおよびHadoopマスターノードに必要となる。



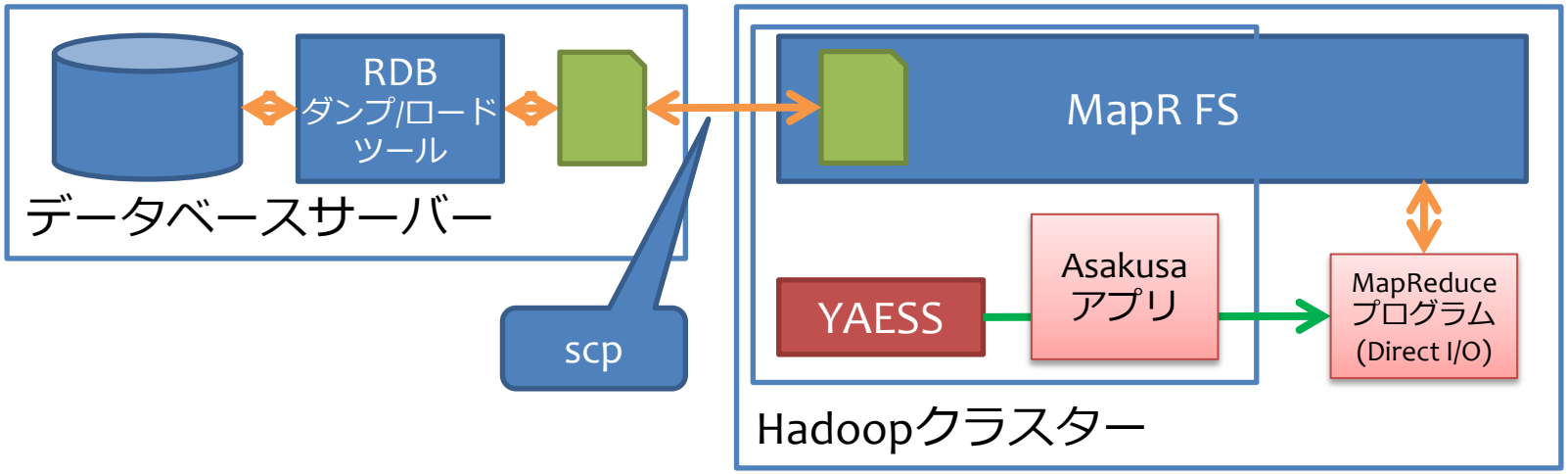
3-3. Direct I/O＋RDBダンプ/ロードツール＋Amazon S3

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、s3cmd等でS3に転送してAsakusaアプリケーションのDirect I/Oで読み込む。
  - AsakusaアプリケーションのDirect I/OでS3上にCSVファイルを作成し、s3cmd等でDBサーバーに転送してRDBロードツールでDBに反映する。
- 構成の特徴
  - CSVファイルをAmazon S3に配置する。
  - DBサーバーにS3アクセスツール（s3cmd等）をインストール・設定する必要がある。
  - DBサーバーおよびHadoopクラスターからS3へアクセス（すなわちインターネット接続）できる必要がある。
  - 大容量のファイルを扱う場合、明示的にファイル分割しておかないと、DBサーバー⇔S3間の転送に影響が出たり、Mapタスクが分散しなかったりする。
  - ファイルを格納できるだけのディスク容量がDBサーバーに必要となる。



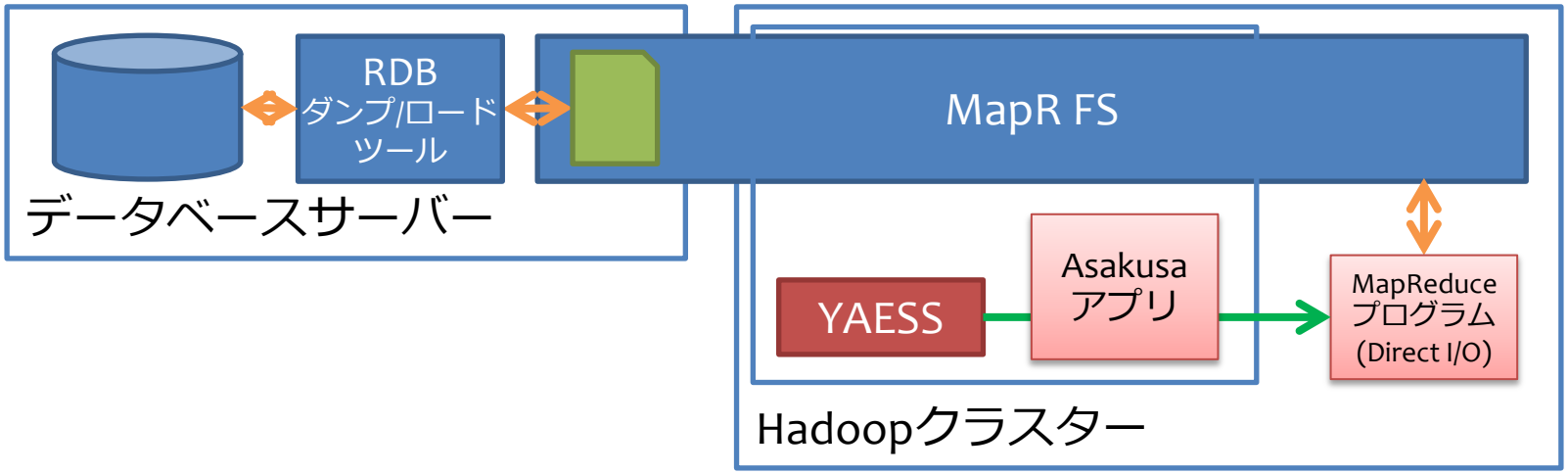
3-4-1. Direct I/O＋RDBダンプ/ロードツール＋転送＋MapR

- 概要
  - RDBダンプツールでDBサーバー上にCSVファイルを作成し、scp等でMapRのNFSに転送してAsakusaアプリケーションのDirect I/Oで読み込む。
  - AsakusaアプリケーションのDirect I/OでMapR FS上にCSVファイルを作成し、scp等でDBサーバーに転送してRDBロードツールでDBに反映する。
- 構成の特徴
  - MapR FSのNFSマウント機能を利用する。
    - MapR FSのNFSマウント領域にファイル転送するだけで直接MapR FS上にファイルが作られる為、3-2-2「Direct I/O＋RDBダンプ/ロードツール＋転送＋HDFS」で行っているような「ローカルファイルシステムからHDFSへ転送する」という処理が不要。
  - HadoopクラスターをMapRで構成し、NFSの設定をする必要がある。
  - DBサーバーとHadoopクラスター間でデータ転送できる必要がある。（scpの場合はSSH接続できる必要がある）
  - ファイルを格納できるだけのディスク容量がDBサーバーに必要となる。



3-4-2. Direct I/O＋RDBダンプ/ロードツール＋MapRクライアント

- 概要
  - DBサーバーでNFSマウントする。そこにCSVファイルを作成し、AsakusaアプリケーションのDirect I/Oで読み書きする。
- 構成の特徴
  - DBサーバー上にMapR FSのNFSマウントポイントを作成する。
    - DBサーバー上のNFSマウント領域にファイルを作るだけで直接MapR FS上にファイルが作られる為、scp等のファイル転送処理が不要。
  - HadoopクラスターをMapRで構成する。また、DBサーバーにMapRクライアントをインストールし、NFSの設定を行う必要がある。
  - MapRクライアントとHadoopクラスター間でMapRのプロトコルの通信が出来るようにする必要がある。

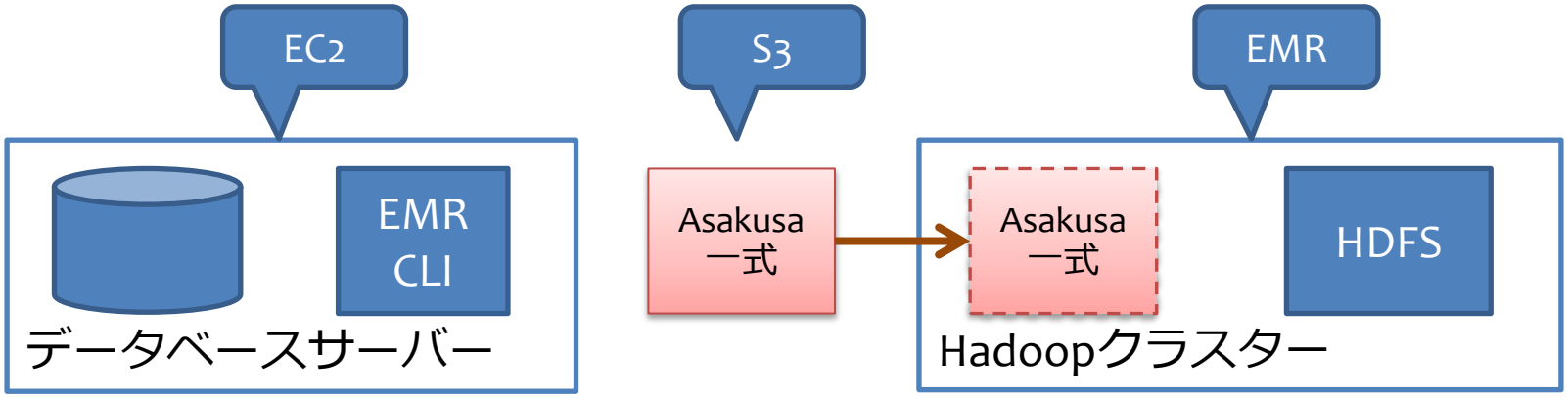


ベンチマークを測定した環境

ベンチマーク測定環境

ベンチマーク測定環境

- 前掲の連携パターンの一部を実際に構築し、実行時間を計測した。
- 環境はAWSを使用した。
  - HadoopクラスターはEMR
  - EC2でデータベースサーバーを構築
    - ここからEMR CLIでEMRの操作を行う。



各種プロダクトのバージョン

- Asakusa Framework 0.4.0
- RDBMS
  - PostgreSQL 9.1
- Hadoop
  - EMR 1.0.3
  - EMR MapR 0.20.205 M3
- Apache Sqoop 1.4

EC2インスタンス（DBサーバー用）の構成

- リージョン：Asia Pacific(Tokyo)
- インスタンスタイプ：m1.xlarge
  - 8ECU・メモリー15GB
  - ストレージ：EBS 70GB
- CentOS6.2
- PostgreSQL9.1
  - DBのダンプはpsqlでCOPY文を実行する。
  - DBへのロードはpg_bulkloadを使用する。
- s3cmd 1.1β
  - Amazon S3のファイルを操作するコマンド。
  - s3cmd 1.0は5GB以上のファイルを扱えない為、1.1を使用する。

EMRの構成

- AMIバージョン：latest
- マスター
  - インスタンスタイプ：m1.xlarge
- スレーブ
  - インスタンスタイプ：m1.xlarge
  - ノード数8

Hadoopのバージョン

- EMRの1.0.3版
- EMR MapRの0.20.205 M3版
  - EMR CLIを使って起動させるコマンドの例

```
elastic-mapreduce --create --alive ¥
--name example-emr ¥
--hadoop-version 0.20.205 ¥
--ami-version latest ¥
--enable-debugging ¥
--master-instance-type m1.xlarge ¥
--slave-instance-type  m1.xlarge ¥
--num-instances 9 ¥
--with-supported-products mapr-m3 ¥
--bootstrap-action s3://elasticmapreduce/bootstrap-actions/run-if ¥
  --args "instance.isMaster=true,s3n://bucketname/example-copy/bootstrap-exa
mple-copy.sh" ¥
--log-uri s3n://bucketname/example-copy/log
```

MapRにするには、
--with-supported-products
を指定する。

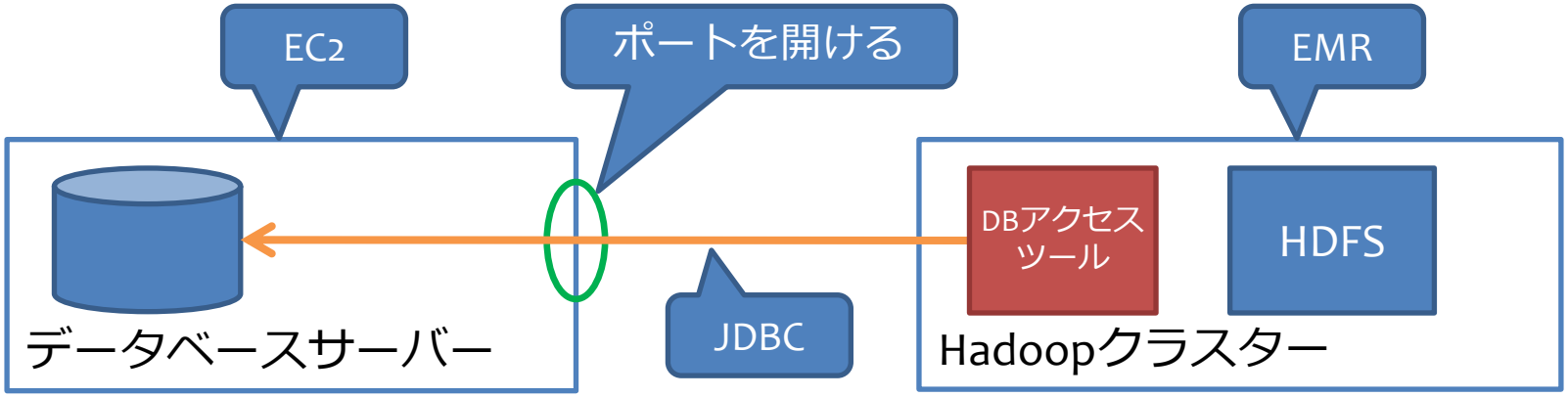
  - 今回は、bootstrap-actionでAsakusa FrameworkやSqoopのインストール・設定を行った。

備考

- Hadoopマスターノードのディスク容量
  - Hadoopのマスターノードのローカルファイルシステム上にファイルを置く構成では、マスターノードのディスク容量に留意する必要がある。
    - デフォルトのEMR
      - /tmpは10GBしか無いが、/mntには400GB以上（インスタンスの種類次第）ある。
    - EMR MapR
      - /mntは/tmpと同じボリュームにマウントされているので、10GBしか無い。
        - » マスターノードのローカルファイルシステム上に大容量ファイルを置くような方式は、EMR MapRの場合は採れない。
      - /mapr/my.cluster.com/にEMRのMapR FSがNFSマウントされており、特別な初期化や設定等をせずに使用できる。

AWSセキュリティーグループの設定

- EMRからEC2のDBへアクセスする構成では、EC2側のDBのポートに対するEMRからのアクセスを許可する必要がある。



Asakusaアプリケーションの概要（１）

- 動作させるAsakusaアプリケーション
  - Asakusa Framework 0.4.0
  - 今回は主にデータ転送にどれくらいの時間がかかるかを見る為、アプリケーションとしては無処理（入力データをそのまま出力するだけ）とする。
  - CSVファイルにしたときに1レコード当たり1kBになるようなデータを使用する。

Asakusaアプリケーションの概要（２）

- WindGate JDBC版の仕様
  - テーブルから読み込み、そのまま同レイアウトの別テーブルへ書き込む。

- WindGate File版の仕様
  - 入力ファイルは１ファイルとする。
    - RDBのダンプでも、PostgreSQLのCOPY文によって１ファイルだけ作成する。
  - 出力ファイルは（Direct I/Oと違ってファイルを分割する機能が無いので）１ファイルとなる。
    - RDBのロードツールではその１ファイルだけを読み込めばよい。

Asakusa FrameworkのWindGateの設定

- $ASAKUSA_HOME/windgate/profile/asakusa.properties

```
# Local File System
resource.local=com.asakusafw.windgate.stream.file.FileResourceProvider
resource.local.basePath=/mnt/tmp/windgate-${USER}

# JDBC
resource.jdbc=com.asakusafw.windgate.jdbc.JdbcResourceProvider
resource.jdbc.driver=org.postgresql.Driver
resource.jdbc.url=jdbc:postgresql://${RDB_HOSTNAME}:5432/asakusa
resource.jdbc.user=asakusa
resource.jdbc.password=asakusa
resource.jdbc.batchGetUnit=1000
resource.jdbc.batchPutUnit=1000
resource.jdbc.connect.retryCount=3
resource.jdbc.connect.retryInterval=10
resource.jdbc.statement.truncate=TRUNCATE TABLE {0}
resource.jdbc.properties.loginTimeout=10
```

EMR用のテンポラリーディレクトリーを指定

DBサーバーはEC2インスタンスなので、IPアドレスが（固定で割り当てない限り）毎回変わる為、環境変数で指定する。

Asakusaアプリケーションの概要（３）

- Direct I/O版の仕様
  - 入力ファイルは１ファイルとする。
    - PostgreSQLのCOPY文によって１ファイルだけ作成し、それをそのままHDFS等に転送する。
  - 出力ファイルは、ファイル名（出力パターン）に「*」を指定して分割する。
    - これにより、Hadoop（MapReduce）が自動的に分割したファイルそのままになる。
      - 「*」でない出力パターンだと、それに従って分割する（あるいは結合する）処理が実行される為、実行速度が遅くなる。
      - Direct I/Oユーザーガイドの『出力ファイル名のパターン』 http://asakusafw.s3.amazonaws.com/documents/0.4.0/release/ja/html/directio/user-guide.html#id85
  - S3上のファイルが相手でも、設定以外は同じ。

Asakusa FrameworkのS3用の設定

- $ASAKUSA_HOME/core/conf/asakusa-resources.xml
  - Direct I/Oユーザーガイドの『Amazon S3での設定例』
    http://asakusafw.s3.amazonaws.com/documents/0.4.0/release/ja/html/directio/user-guide.html#id40

```
<property>
    <name>com.asakusafw.directio.s3</name>
    <value>com.asakusafw.runtime.directio.hadoop.HadoopDataSource</value>
</property>
<property>
    <name>com.asakusafw.directio.s3.path</name>
    <value>/s3</value>
</property>
<property>
    <name>com.asakusafw.directio.s3.fs.path</name>
    <value>s3n://bucketname/example-copy/data</value>
</property>
```

```
<property>
    <name>com.asakusafw.directio.s3.fragment.min</name>
    <value>-1</value>
</property>
<property>
    <name>com.asakusafw.directio.s3.output.staging</name>
    <value>false</value>
</property>
<property>
    <name>com.asakusafw.directio.s3.output.streaming</name>
    <value>false</value>
</property>
<property>
    <name>com.asakusafw.output.system.dir</name>
    <value>s3n://bucketname/example-copy/system</value>
</property>
<property>
    <name>com.asakusafw.output.local.tempdir</name>
    <value>/mnt/var/asakusa-directio</value>
</property>
```

– この場合、Description側は以下のようになる。

```
public class PosItemFromS3Csv extends AbstractPosItemDioCsvInputDescription
{

    @Override
    public String getBasePath() {
        return "s3/copy-input/csv";
    }

    @Override
    public String getResourcePattern() {
        return "*.csv";
    }

    @Override
    public DataSize getDataSize() {
        return DataSize.LARGE;
    }
}
```

getBasePath()の先頭の文字列「s3」と、asakusa-resources.xmlで指定したcom.asakusafw.directio.s3.pathの値「/s3」が一致するようにする。すると「s3n://bucketname/example-copy/data/copy-input/csv/*.csv」を指すことになる。

Sqoopのバージョン

- Apache Sqoop 1.4
  - RDBMSとHadoopを連携するツール。
    - RDBMS連携ツールは色々あるが、今回はHadoopと同じApacheプロジェクトであるSqoopを使用した。
  - Apache Sqoop 1.3はダウンロードサイトが見つからなかったので、1.4を使用。
    - なお、CDH3版SqoopはEMRのHadoopに対応していない。
  - 動作モードはデフォルト値を使用。
    - Mapタスク数は4
    - ダイレクトモード（RDBMS固有の機能を使用するモード）は使用しない

各パターンの実行時間の測定結果

測定結果

測定対象の構成パターン

- 以下のパターンを測定した。

	構成パターン名	略称	備考
1-1	WindGate JDBC	wgate-jdbc	
3-1-3	WindGate File＋RDBダンプ/ロードツール＋転送	wgate-csv	
2-1	Direct I/O＋RDBMS連携ツール	dio-sqoop-csv	Sqoop CSVファイル
		dio-sqoop-seq	Sqoop SequenceFile
3-2-2	Direct I/O＋RDBダンプ/ロードツール＋転送＋HDFS	dio-fs	
3-3	Direct I/O＋RDBダンプ/ロードツール＋Amazon S3	dio-s3	
3-4-1	Direct I/O＋RDBダンプ/ロードツール＋転送＋MapR	dio-fs-mapr	

  - RDBMS連携ツールはSqoopを使用。SqoopはHDFS上のファイルとしてCSVファイルとSequenceFileの二種類が扱える。

測定対象の構成パターンの選定理由

- DBサーバーにHadoopをインストールする必要があるパターンは除外した。
  - 今回はAmazon EC2でDBサーバーを構築しており、HadoopはAmazon EMRを使っている。
    DBサーバーをHadoopクライアントとして使用する為にはHadoopをインストールする必要があるが、EMRは独自バージョンのHadoopである為、EMR以外のサーバーにインストールすることが出来ない。
    - 他バージョンのHadoopをインストールしても、EMRにアクセスできない。（プロトコルのバージョンが違う為か、エラーになる）

構成パターンの内訳

略称	import（RDB→HDFS）	Asakusaアプリ	export（HDFS→RDB）
wgate-jdbc	なし	WindGate	なし
wgate-csv	psql＋scp put	WindGate	scp get＋pg_bulkload
dio-sqoop-csv	sqoop import(CSV)	Direct I/O	sqoop export(CSV)
dio-sqoop-seq	sqoop import(SeqFile)	Direct I/O	sqoop export(SeqFile)
dio-fs	psql＋scp put＋fs put	Direct I/O	fs get＋scp get＋pg_bulkload
dio-s3	psql＋s3cmd put	Direct I/O	s3cmd get＋pg_bulkload
dio-fs-mapr	psql＋scp put(MapR)	Direct I/O	scp get(MapR)＋pg_bulkload

- import/exportは、WindGateのCSVファイルではローカルディスク、Direct I/OはHDFS・MapR FSやS3が対象

構成パターンの内訳の見方

- 例えばwgate-csvのimportは「psql＋scp put」になっている。
  これは、psql（PostgreSQLのダンプ）とscp put（scpによる転送）の2種類の操作で成り立っていることを意味する。
- 次ページから、それぞれの操作にかかった時間を掲示する。
  最後にパターン毎の実行時間（内訳を合算したもの）を掲示する。

実行時間：RDBダンプ/ロードツール

データサイズ	psql	pg_bulkload
100MB	3	4
500MB	19	18
1GB	33	54
5GB	184	214
10GB	362	440

秒

- psql
  - PostgreSQLからCOPY文によるCSVファイル作成（ダンプ）
- pg_bulkload
  - CSVファイルのPostgreSQLへのバルクロード

実行時間：scp put/get

データサイズ	scp put	scp get	scp put (MapR)	scp get (MapR)
100MB	3	2	3	3
500MB	11	10	14	11
1GB	20	20	25	21
5GB	96	100	108	106
10GB	207	244	210	205

秒

- scp put
  - DBサーバーからマスターノードへのファイル転送
- scp get
  - マスターノードからDBサーバーへのファイル転送

実行時間：hadoop fs -put/-get

データサイズ	fs -put	fs -get
100MB	5	3
500MB	11	8
1GB	18	13
5GB	71	53
10GB	141	122

秒

- hadoop fs -put
  - マスターノードのローカルからHDFSへの転送
- hadoop fs -get
  - HDFSからマスターノードのローカルへの転送

実行時間：s3cmd put/get

データサイズ	s3cmd put	s3cmd get
100MB	7	3
500MB	45	14
1GB	89	27
5GB	366	141
10GB	797	292

秒

- s3cmd put
  - DBサーバーからAmazon S3へのファイル転送
- s3cmd get
  - Amazon S3からDBサーバーへのファイル転送

実行時間：Sqoop（RDBMS連携ツール）

データサイズ	import（CSV）	export（CSV）	import（SeqFile）	export（SeqFile）
100MB	37	40	33	45
500MB	41	76	39	89
1GB	48	127	47	128
5GB	104	510	99	497
10GB	461	954	468	958

秒

- sqoop import
  - DBを読み込んでHDFSにファイルを作成
- sqoop export
  - HDFSのファイルを読み込んでDBへ書き込む

実行時間：Asakusaアプリケーション

秒

データサイズ	wgate jdbc	wgate csv	dio csv	dio seq	dio s3	wgate jdbc (MapR)	dio csv (MapR)
100MB	66	53	29	32	49	40	22
500MB	160	103	35	38	92	113	27
1GB	267	154	40	42	130	204	29
5GB	1127	510	64	76	477	921	46
10GB	2255	990	85	98	895	1845	62

- これは、YAESSによって起動されて終了するまでの時間。

実行時間：Asakusaアプリケーション（内訳）

データサイズ	wgate jdbc 入力	wgate jdbc 出力	wgate jdbc (MapR) 入力	wgate jdbc (MapR) 出力
100MB	7	54	6	30
500MB	24	130	18	90
1GB	45	216	33	166
5GB	202	921	156	760
10GB	442	1802	357	1483

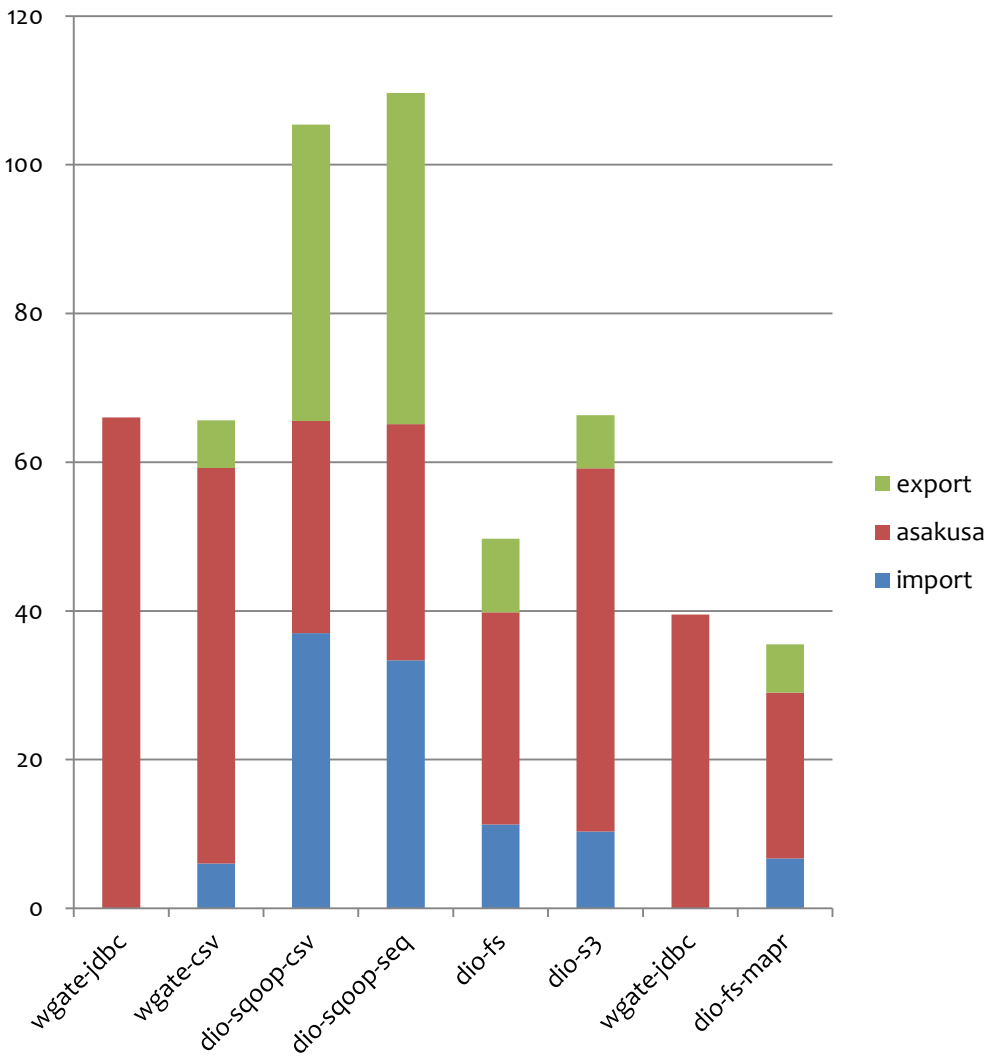
秒

- Asakusaアプリケーションもprologue＋importフェーズ・epilogue＋exportフェーズが外部入出力に相当する。wgate-jdbcでは内訳は上記のようになっていた。
  - YEASSユーザーガイドの『フェーズ』
    http://asakusafw.s3.amazonaws.com/documents/0.4.0/release/ja/html/yaess/user-guide.html#id2

実行時間　100MB

略称	import	Asakusa	export	全体
wgate-jdbc		66		66
wgate-csv	6	53	6	66
dio-sqoop-csv	37	29	40	105
dio-sqoop-seq	33	32	45	110
dio-fs	11	29	10	50
dio-s3	10	49	7	66
wgate-jdbc		40		40
dio-fs-mapr	7	22	6	36

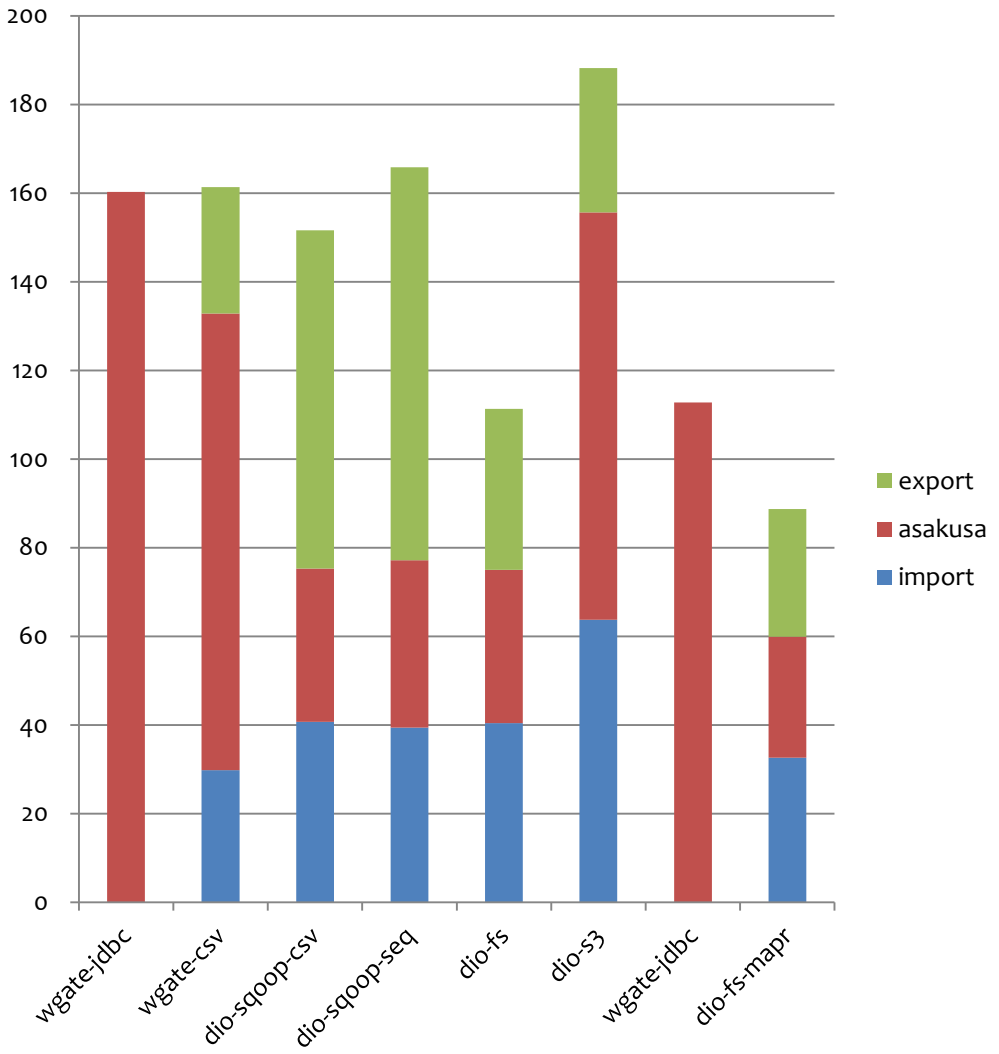
- 下2行はMapR



実行時間　500MB

略称	import	Asakusa	export	全体
wgate-jdbc		160		160
wgate-csv	30	103	29	161
dio-sqoop-csv	41	35	76	152
dio-sqoop-seq	39	38	89	166
dio-fs	40	35	36	111
dio-s3	64	92	33	188
wgate-jdbc		113		113
dio-fs-mapr	33	27	29	89

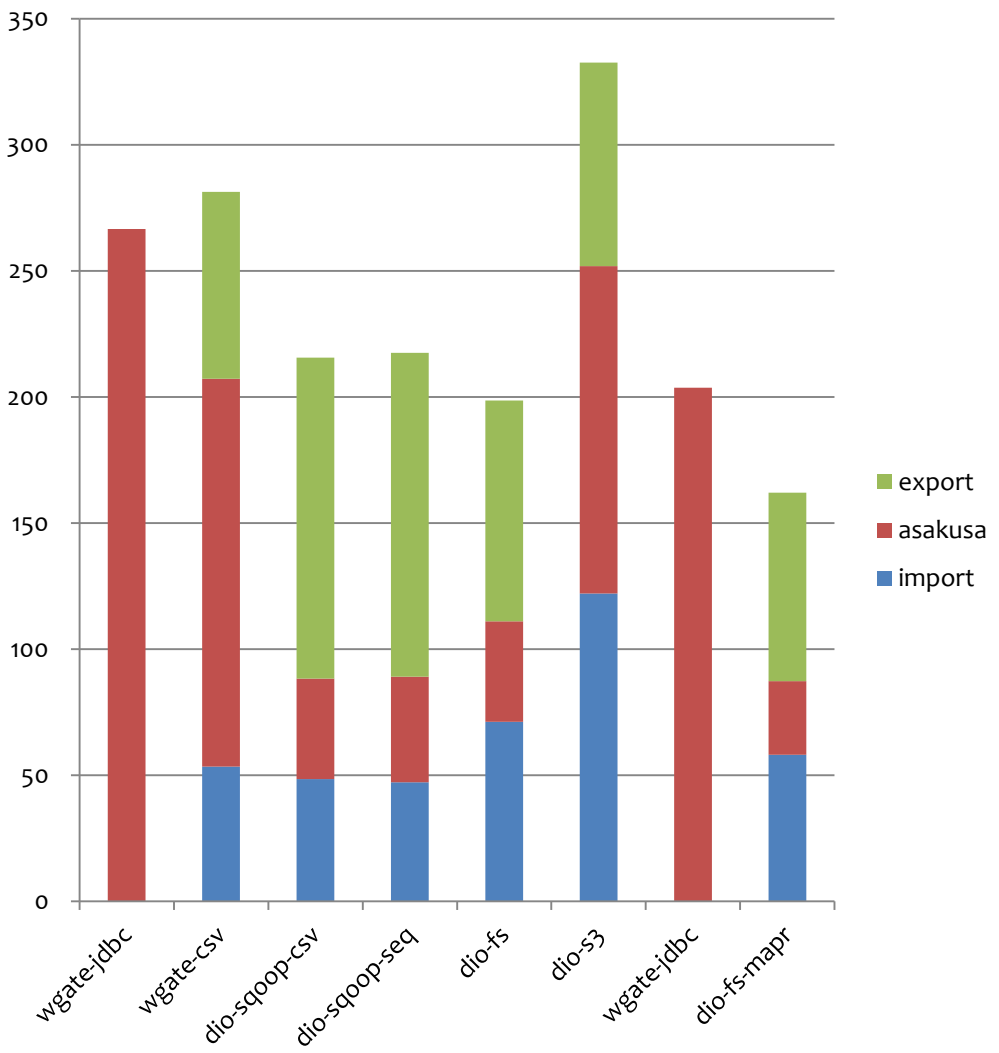
- 下2行はMapR



実行時間　1GB

略称	import	Asakusa	export	全体
wgate-jdbc		267		267
wgate-csv	54	154	74	281
dio-sqoop-csv	48	40	127	216
dio-sqoop-seq	47	42	128	218
dio-fs	71	40	88	199
dio-s3	122	130	81	333
wgate-jdbc		204		204
dio-fs-mapr	58	29	75	162

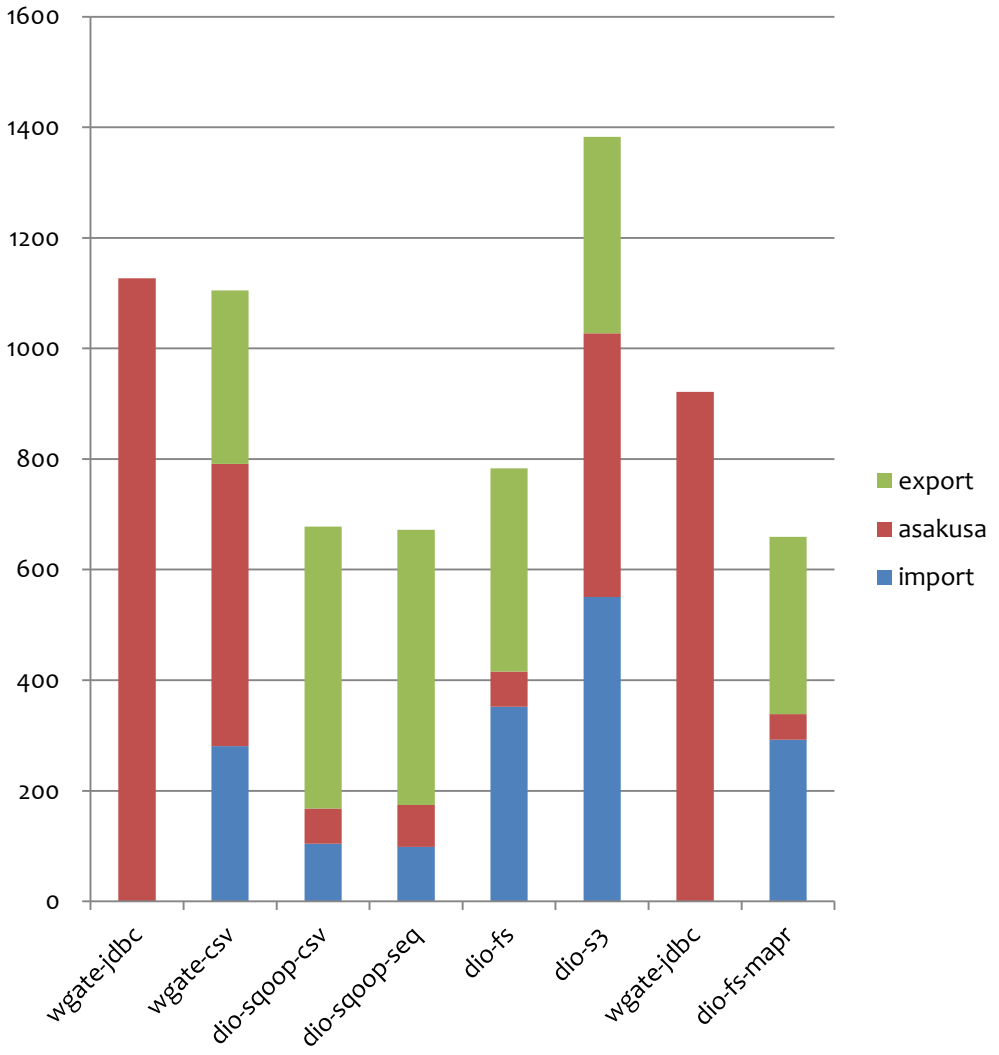
- 下2行はMapR



実行時間　5GB

略称	import	Asakusa	export	全体
wgate-jdbc		1127		1127
wgate-csv	281	510	314	1105
dio-sqoop-csv	104	64	510	678
dio-sqoop-seq	99	76	497	672
dio-fs	352	64	367	783
dio-s3	550	477	355	1383
wgate-jdbc		921		921
dio-fs-mapr	292	46	321	659

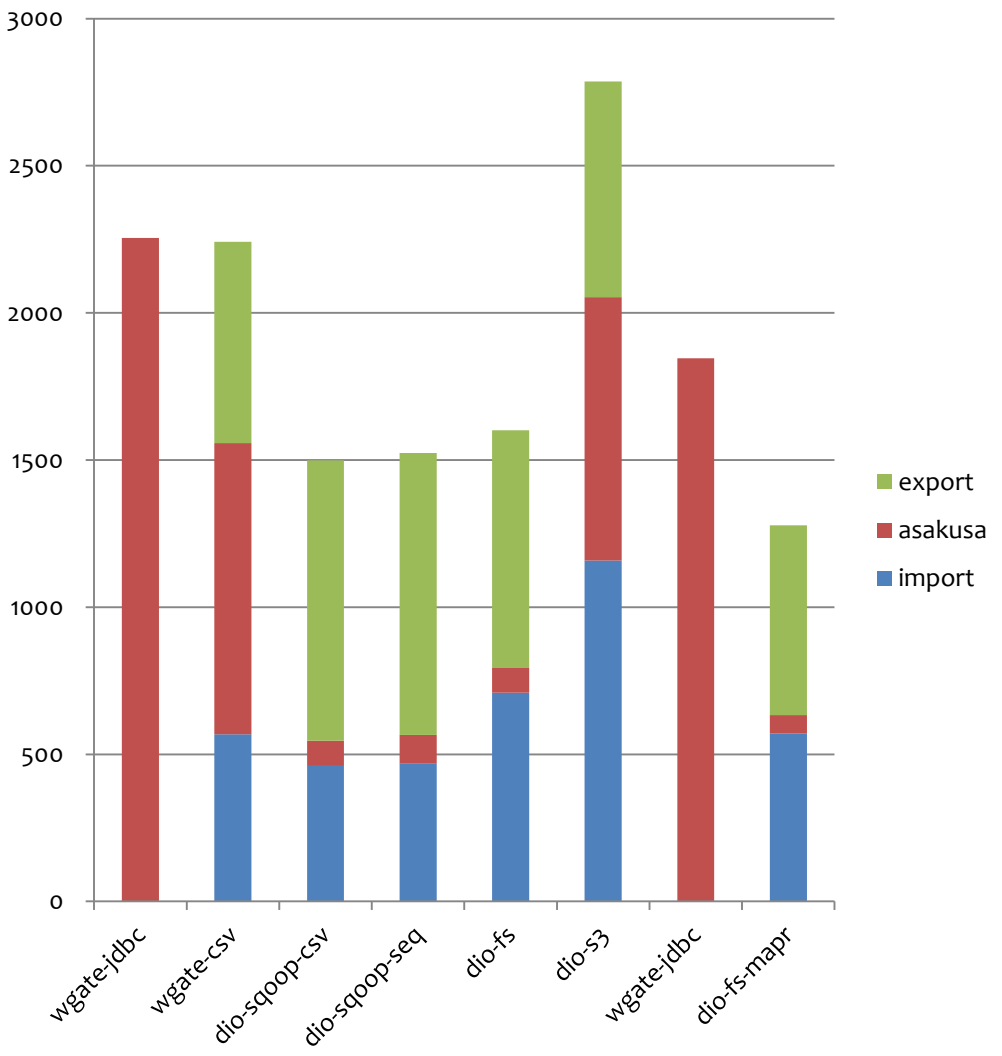
- 下2行はMapR



実行時間　10GB

略称	import	Asakusa	export	全体
wgate-jdbc		2255		2255
wgate-csv	568	990	684	2242
dio-sqoop-csv	461	85	954	1499
dio-sqoop-seq	468	98	958	1524
dio-fs	710	85	806	1601
dio-s3	1158	895	733	2786
wgate-jdbc		1845		1845
dio-fs-mapr	571	62	645	1278

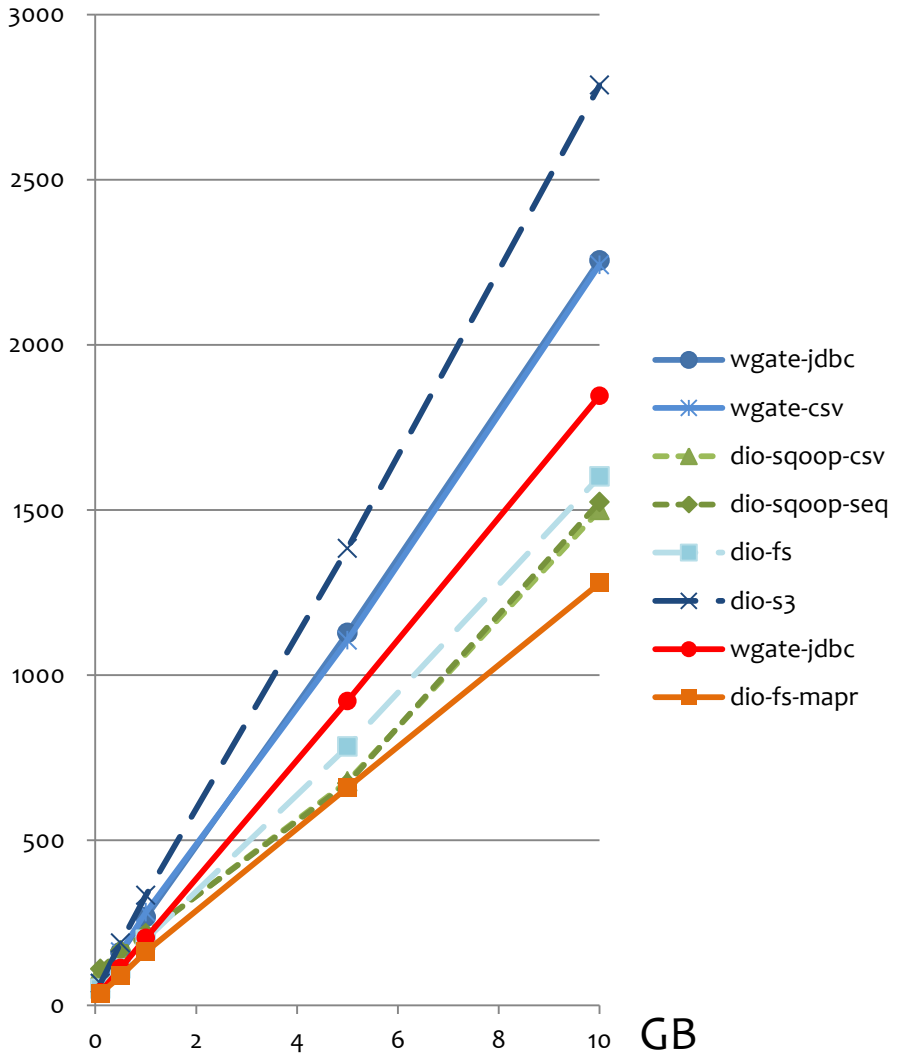
- 下2行はMapR



実行時間（全体）

略称	100M	500M	1G	5G	10G
wgate-jdbc	66	160	267	1127	2255
wgate-csv	66	161	281	1105	2242
dio-sqoop-csv	105	152	216	678	1499
dio-sqoop-seq	110	166	218	672	1524
dio-fs	50	111	199	783	1601
dio-s3	66	188	333	1383	2786
wgate-jdbc	40	113	204	921	1845
dio-fs-mapr	36	89	162	659	1278

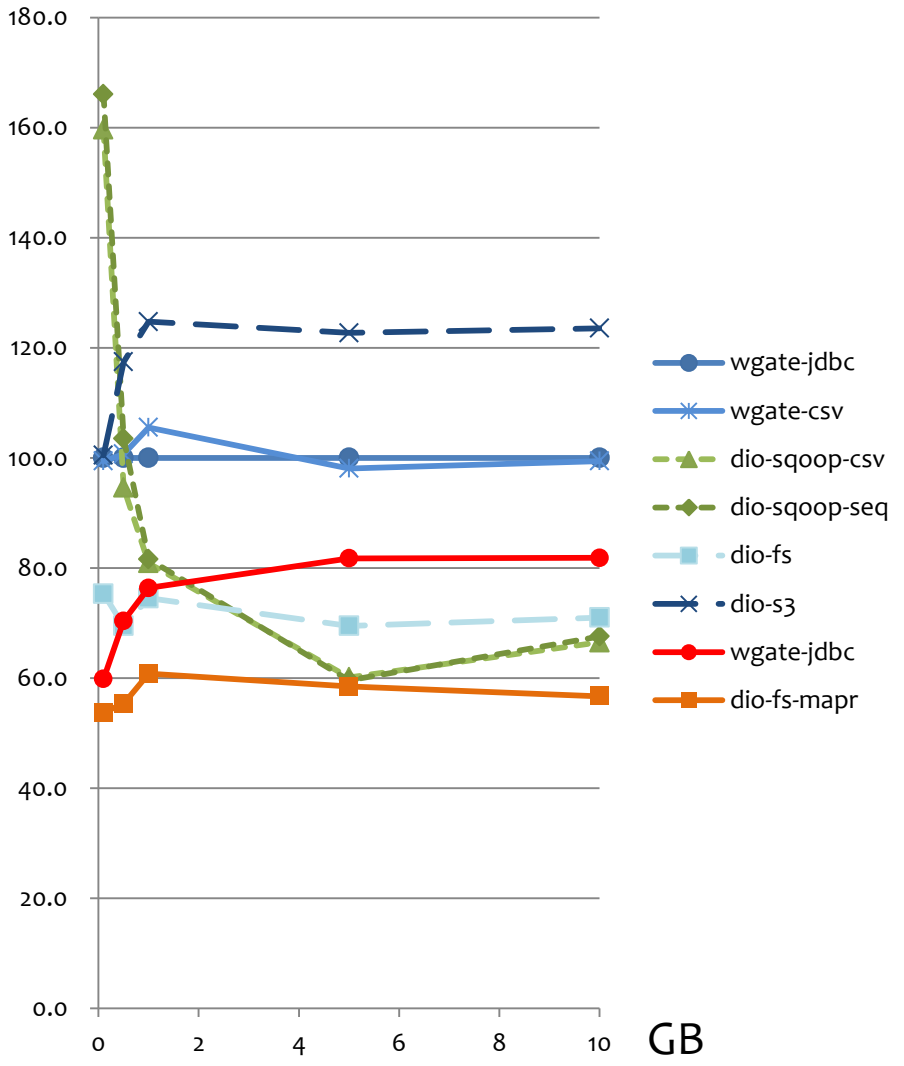
- 下2行はMapR



実行時間の比

略称	100M	500M	1G	5G	10G
wgate-jdbc	100.0	100.0	100.0	100.0	100.0
wgate-csv	99.4	100.7	105.5	98.1	99.4
dio-sqoop-csv	159.7	94.6	80.9	60.1	66.5
dio-sqoop-seq	166.1	103.5	81.6	59.6	67.6
dio-fs	75.3	69.5	74.5	69.5	71.0
dio-s3	100.5	117.4	124.7	122.7	123.5
wgate-jdbc	59.8	70.4	76.4	81.8	81.8
dio-fs-mapr	53.8	55.4	60.8	58.5	56.7

- wgate-jdbcとの実行時間の比較
  （自分÷wgate-jdbc×100）
  （数値が小さい方が速い）
- 下2行はMapR



AWS環境で測定した実行時間に関する考察

まとめ

まとめ

- どのパターンが速いかは、扱うデータ量によって異なる。
  - 概ね、Direct I/OとRDBMS連携ツール（今回はSqoop）を組み合わせて使う方がWindGateより速かった。
  - 今回は入力と出力が同サイズだったが、S3以外のパターンでは概ねexportの時間の方が長かった。集計等を行って出力件数が減るようなアプリケーションなら、入力と出力で方式を変えると全体の実行時間が速くなるかもしれない。
- MapR FSを使用すると、同じコンポーネント構成でもHDFSより速かった。

- Asakusa Framework関連
  - 1-1「WindGate JDBC」は今回のパターンの中では遅い部類になった。
    - パラメーターは初期値のまま（WindGateプロファイルでコミット件数等のコメントアウトを外した状態）だったので、それらを変えれば速くなる可能性はある。
  - 3-1-3「WindGate File＋RDBダンプ/ロードツール＋転送」は1-1「WindGate JDBC」とほぼ同じ速度だった。
    - scp転送しない3-1-1ならもう少し速くなると思われる。
  - Direct I/OとRDBMS連携ツール（Sqoop）の組み合わせはデータ量が多いとWindGateより速かった。
    - Sqoop以外でも性能の良いRDBMS連携ツールがあれば、Direct I/Oと組み合わせて有効に使えると思われる。

- Apache Sqoop関連
  - 今回の測定では、データ量が多いと他の方式より速かった。
  - CSVファイルとSequenceFileとでは、実行時間に大きな差は無かった。
  - 今回はデフォルト設定で実行した為、ダイレクトモード（RDBMS固有機能を使用）やタスク数等のチューニングを行うともっと速くなる可能性がある。

- Amazon S3関連
  - 今回の測定では、S3上のファイルに直接アクセスする方式は他のパターンに比べて遅かった。
  - 今回はAsakusaアプリケーションがS3から読み込む入力ファイルは１つだったが、明示的に分割すれば速くなる可能性がある。
    - Asakusa Frameworkの実行基盤であるHadoopは、S3のファイルは自動では分散処理しない為。
  - 今回はファイルを分割してS3へ分散してアップロードすることはしなかったが、それが出来ればもっと速くなる可能性がある。

- MapR関連
  - 今回の測定では、MapR FSはHDFSより高速だった。
    - WindGate JDBCも一時ファイルを分散FS上に作っているので、MapRを使う方が速くなる。

MapR FSのNFSマウント領域にscp転送するだけで分散FS上にファイルを置けるのも便利である。

ただし、現在のところ、Amazon VPC内のEMRではMapRを使用できない。