Amazon EMR のベストプラクティス

Parviz Deyhim

2013 年 8 月

（本ペーパーの最新版については、http://aws.amazon.com/whitepapers/ をご覧ください）

目次

要約

アマゾン ウェブ サービス（AWS）クラウドは、ビッグデータ解析を高速化します。瞬時の拡張性と伸縮自在性を備えているため、インフラストラクチャにわずらわされることなく、解析に集中できます。AWS には、大きなデータセットのインデックス作成、大量の科学データの解析、クリックストリームログの処理など、大量のデータを扱う事実上すべてのプロジェクトに使用できるビッグデータツールおよびサービスが用意されています。

その 1 つ、Amazon Elastic MapReduce（EMR）は、Amazon Elastic Compute Cloud（EC2）の上に完全に管理されたホスト型の Hadoop フレームワークを提供します。本ペーパーでは、AWS へデータを移動し、そのデータを収集および集約するベストプラクティスに注目するとともに、処理を高速化するために Amazon EMR クラスターでセットアップおよび設定する一般的なアーキテクチャパターンについて説明します。また、信頼性のある方法で高いスループットおよび低コストで大量のデータを処理できるよう、パフォーマンスおよびコストを最適化するための手法についてもいくつか説明します。

はじめに

ビッグデータは、大量のデータの収集、格納、処理、視覚化であり、企業はそこから知識を抽出し、その知識からビジネスにおける貴重な洞察を導き出し、ビジネスにおける決定を適切かつ迅速に下すことができます。データ解析プラットフォームにおける主な課題としては、インストールおよび運用管理、変動する負荷に合わせたデータ処理容量の動的な割り当て、包括的な解析のための複数のソースからのデータの集約などが挙げられます。Open Source Apache Hadoop およびそのツールのエコシステムではこうした問題の解決を支援します。Hadoop はデータボリュームの増大に合わせ水平的に拡張できるとともに、構造化されていないデータと構造化されているデータを同じ環境で処理することができます。

Amazon Elastic MapReduce（Amazon EMR）は、AWS での Hadoop および関連するビッグデータアプリケーションの実行を簡素化します。 Hadoop インストールを管理するためのコストおよび複雑な作業を解消します。つまり、開発者や企業は大きな設備投資をかけずに解析を行うことができます。今日、AWS クラウド内の最適なパフォーマンスを実現する Hadoop クラスターを最新のハイパフォーマンスコンピューティングハードウェアおよびネットワークで数分以内で立ち上げることができ、こうしたハードウェアを購入するための設備投資をかける必要はありません。オンデマンドで実行中のクラスターを拡大および縮小できます。つまり、質問に対する迅速な回答が必要であれば、クラスターのサイズを拡大し、データをより迅速に処理することができます。Hadoop の MapReduce アーキテクチャを使用して AWS クラウドで実行中の仮想サーバークラスターに演算作業を分散させることにより、大量のデータを解析および処理できます。

データ処理だけでなく、大量のデータの解析でも、データの収集、移行、最適化が必要です。

図 1: データフロー

本ホワイトペーパーでは、AWS へデータを移動するためのベストプラクティス、データを収集、圧縮、集約するための戦略、および Amazon EMR クラスターを処理するためのセットアップおよび設定する一般的なアーキテクチャパターンについて説明します。また、コストを最適化するための例、およびリザーブドインスタンスやスポットインスタンスなどさまざまな Amazon EC2 購入オプションを使用する例も提供します。本ペーパーは、Amazon EMR および Apache Hadoop に関する概念を理解し、多少の経験を有している方々を対象読者としています。Amazon EMR の概要については、Amazon EMR 開発者ガイド [1] を参照してください。Hadoop の概要については、Hadoop: The Definitive Guide [2]を参照してください。

AWS へのデータの移動

現在ご使用のストレージから Amazon Simple Storage Service（Amazon S3）へ、または Amazon S3 から Amazon EMR および Hadoop Distributed File System（HDFS）に大量のデータを移動する方法には、さまざまなアプローチがあります。ただし、いずれのアプローチを取る場合も、確保できるデータ帯域幅を戦略的に使うことが不可欠です。適切に最適化すれば、1 日あたり数テラバイトのアップロードが可能となります。このような高スループットは、複数のクライアントから並列で AWS にデータをアップロードすることで可能ですが、それには個々のクライアントでマルチスレッディングにより同時アップロードを実現する方法と、マルチパートアップロードを採用してさらなる並列処理を図る方法があります。ウィンドウスケーリング [3]や選択的送達確認 [4] などの TCP 設定を調整し、さらにデータスループットを向上させることができます。以下のシナリオでは、使用可能なスループットを十分に利用することにより、カレントのローカルストレージの場所（データセンター）から AWS へのデータ移行を最適化するための 3 つの方法について説明します。

シナリオ 1: HDFS（データセンター）から Amazon S3 への大量のデータの移動

2 つのツール、S3DistCp と DistCp で、ローカルな（データセンター）HDFS ストレージに格納されているデータを Amazon S3 に簡単に移動できます。Amazon S3 は、その高い耐久性とセキュリティやライフサイクル管理などのエンタープライズクラスの機能を備えた、構造化されていないデータファイル向けの優れた永続的なストレージオプションです。

S3DistCp の使用

S3DistCp は DistCp を拡張したもので、AWS、特に Amazon S3 で作業するための最適化機能を備えています。S3DistCp をジョブフローの 1 ステップとして追加することにより、Amazon S3 から HDFS に大量のデータを効率的にコピーでき、EMR クラスターの以降のステップでデータを処理できます。また、S3DistCp を使用して、Amazon S3 バケット間で、または HDFS から Amazon S3 にデータをコピーすることもできます。

S3DistCp は、DistCp と類似した分散化された map-reduce ジョブジョブを使用してデータをコピーします。S3DistCp はマッパーを実行して、送信先にコピーするファイルのリストをコンパイルします。マッパーがファイルのリストのコンパイルを完了すると、リデューサーが実際のデータコピーを実行します。DistCp とは異なり S3DistCp が提供する主な最適化では、リデューサーに複数の HTTP アップロードスレッドを実行させることで、ファイルを並列でアップロードします。

[1] http://aws.amazon.com/documentation/elasticmapreduce/
[2] http://www.amazon.com/Hadoop-Definitive-Guide-Tom-White/dp/1449311520/
[3] http://docs.aws.amazon.com/AmazonS3/latest/dev/TCPWindowScaling.html
[4] http://docs.aws.amazon.com/AmazonS3/latest/dev/TCPSelectiveAcknowledgement.html

S3DistCP を使用することの利点を明らかにするために、S3DistCp と DistCp の対照比較を実施しました。このテストでは、ヴァージニアの Amazon Elastic Compute Cloud（EC2）で実行中の Hadoop クラスターから 50 GB のデータをコピーし、このデータをオレゴンの Amazon S3 バケットにコピーします。このテストは、特定の状況下での S3DistCp と DistCp のパフォーマンスの差を示すものであり、実際の環境では結果が異なる可能性があります。

方法	コピーするデータのサイズ	合計時間
DistCp	50 GB	26 分
S3DistCp	50 GB	19 分

図 2: DistCp と S3DistCp のパフォーマンスの比較

Hadoop クラスターから Amazon S3 に S3DistCp を使用してデータをコピーするには

以下は、お使いの Hadoop インストール環境で S3DistCp を実行して、HDFS から Amazon S3 にデータをコピーする方法の例です。1) Apache Hadoop 1.0.3 ディストリビューション 2) Amazon EMR AMI 2.4.1 で以下のステップをテストしました。それ以外の Hadoop ディストリビューションではこのプロセスをテストしていないため、ここで示した Hadoop ディストリビューション（Apache Hadoop 1.0.3）以降でまったく同じステップが機能するかは保証できません。

1. 小さな Amazon EMR クラスター（単一ノード）を起動します。

```
elastic-mapreduce --create --alive --instance-count 1 --instance-type m1.small --
ami-version 2.4.1
```

2. Amazon EMR のマスターノード（/home/Hadoop/lib）から Hadoop インストールパス（例: /usr/local/hadoop/lib）の /lib ディレクトリ下のローカルな Hadoop マスターノードに以下の jar をコピーします。これらの jar があるかどうかは、Hadoop のインストール状況に応じて異なります。Apache Hadoop ディストリビューションにはこれらの jar はありません。

```
/home/hadoop/lib/emr-s3distcp-1.0.jar
/home/hadoop/lib/aws-java-sdk-1.3.26.jar
/home/hadoop/lib/guava-13.0.1.jar
/home/hadoop/lib/gson-2.1.jar
/home/hadoop/lib/EmrMetrics-1.0.jar
/home/hadoop/lib/protobuf-java-2.4.1.jar
/home/hadoop/lib/httpcore-4.1.jar
/home/hadoop/lib/httpclient-4.1.1.jar
```

3. `core-site.xml` ファイルを編集して、AWS 認証情報を挿入します。次に、`core-site.xml` 構成ファイルを Hadoop クラスターノードすべてにコピーします。ファイルのコピー後、変更を有効にするためにサービスまたはデーモンを再起動する必要はありません。

```
<property>
  <name>fs.s3.awsSecretAccessKey</name>
  <value>YOUR_SECRETACCESSKEY</value>
</property>

<property>
  <name>fs.s3.awsAccessKeyId</name>
  <value>YOUR_ACCESSKEY</value>
```

```
  </property>
  <property>
    <name>fs.s3n.awsSecretAccessKey</name>
    <value>YOUR_SECRETACCESSKEY</value>
  </property>

  <property>
    <name>fs.s3n.awsAccessKeyId</name>
    <value>YOUR_ACCESSKEY</value>
  </property>
```

4. 以下の例を使用して s3distcp を実行します（HDFS_PATH、YOUR_S3_BUCKET、PATH を変更します）。

```
hadoop jar /usr/local/hadoop/lib/emr-s3distcp-1.0.jar -libjars
/usr/local/hadoop/lib/gson-2.1.jar,/usr/local/hadoop/lib/guava-
13.0.1.jar,/usr/local/hadoop/lib/aws-java-sdk-1.3.26.jar,/usr/local/hadoop/lib/emr-
s3distcp-1.0.jar,/usr/local/hadoop/lib/EmrMetrics-
1.0.jar,/usr/local/hadoop/lib/protobuf-java-
2.4.1.jar,/usr/local/hadoop/lib/httpcore-4.1.jar,/usr/local/hadoop/lib/httpclient-
4.1.1.jar --src HDFS_PATH --dest s3://YOUR_S3_BUCKET/PATH/ --disableMultipartUpload
```

DistCp の使用

DistCp（distributed copy）は、クラスター間またはクラスター内の大きなデータのコピーに使用するツールです。Amazon EMR を使用して、レポートの他、ディストリビューション、エラー処理、リカバリを行います。ファイルとディレクトリのリストをマップタスクの入力に展開します。それぞれのタスクがソースリストに指定されているファイルの一部をコピーします。

DistCp は S3DistCp と同じような分散方法で HDFS から Amazon S3 にデータをコピーできますが、DistCp はそれほど高速ではありません。DistCp では以下のアルゴリズムを使用して、必要なマッパー数を算出します。

*min (total_bytes / bytes.per.map, 20 * num_task_trackers)*

通常はこの計算式で問題ありませんが、適切なデータ量が算出されない場合もあります。DistCp を使用し、データをコピーするために使用するマッパーの数がクラスターの合計マッパー容量よりも小さい場合、`-m number_of_mappers` オプションを指定することにより、DistCp がファイルをコピーするために使用するマッパーの数を増やすことができます。

以下は、DistCp コマンドが HDFS 上の/data ディレクトリを特定の Amazon S3 バケットにコピーする例です:

```
hadoop distcp hdfs:///data/ s3n://awsaccesskey:awssecrectkey@somebucket/mydata/
```

DistCp の使用の詳細およびチュートリアルについては、http://hadoop.apache.org/docs/r0.19.2/distcp.html を参照してください。

シナリオ 2: ローカルディスク（非 HDFS）から Amazon S3 への大量のデータの移動

シナリオ 1 では、分散コピーツール（DistCp および S3DistCp）を使用して並列で AWS にデータをコピーする方法について説明しました。シナリオ 1 で実現する並列処理は、複数の HDFS ノードにデータが格納され、複数のノードが同時にデータをコピーするため、可能でした。幸い、HDFS を使用しない場合でも、データを効率的に移動するための方法がいくつかあります。

Jets3t Java ライブラリの使用

JetS3t は、開発者が Amazon S3 または Amazon CloudFront とやり取りする強力でありながらシンプルなアプリケーションを作成するためのオープンソースの Java ツールキットです。JetS3t は低レベル API を提供しますが、Java アプリケーションを作成することなく Amazon S3 または Amazon CloudFront を使用するためのツールも付属しています。

JetS3t ツールキットで提供されるツールの 1 つが Synchronize というアプリケーションです。Synchronize は、コンピュータ上のディレクトリを Amazon S3 バケットと同期化するためのコマンドラインアプリケーションです。バックアップや異なるコンピュータ間でファイルの同期化を行うのに理想的です。

Synchronize の利点の 1 つは設定の柔軟性です。できるだけ多くのアップロードスレッドを開くように、Synchronize を設定できます。この柔軟性により、使用可能な帯域幅をすべて使用し、使用可能なスループットを十分活用できます。

Synchronize をセットアップするには

1. 以下の URL: http://jets3t.s3.amazonaws.com/downloads.html から Jets3t をダウンロードします。
2. `jets3t` を解凍します。
3. `synchronize.properties` ファイルを作成し、`accesskey` および `secretkey` の値を自分の AWS アクセスキー識別子に置き換え、以下のパラメータを追加します:

```
accesskey=xxx
secretkey=yyy
upload.transformed-files-batch-size=100
httpclient.max-connections=100
storage-service.admin-max-thread-count=100
storage-service.max-thread-count=10
threaded-service.max-thread-count=15
```

4. 以下のコマンドライン例を使用して、Synchronize を実行します:

```
bin/synchronize.sh -k UP somes3bucket/data /data/ --properties
synchronize.properties
```

GNU Parallel の使用

GNU Parallel は、1 つまたは複数のコンピュータを使用してジョブを並列で実行するためのシェルツールです。GNU パラレル は、入力の各ラインに対し実行する単一のコマンドまたは小さいスクリプトのジョブを実行します。GNU Parallel を使用することで、複数のスレッドを同時に開き、複数のファイルをアップロードするプロセスを並列処理できます。一般的には、使用可能な帯域幅の大半を使用するために、できるだけ多くのパラレルアップロードスレッドを開きます。

GNU Parallel の使用例を以下に示します:

1. Amazon S3 にアップロードするファイルのリストをカレントのフルパスを使用して作成します。
2. GNU Parallel を Amazon S3 のいずれかのアップロード/ダウンロードツールと一緒に、以下のコマンドライン例を使用してできるだけ多くのスレッドで実行します:

```
ls | parallel -j0 -N2 s3cmd put {1} s3://somes3bucket/dir1/
```

上記の例では、カレントディレクトリ（`ls`）の内容をコピーし、`s3cmd` コマンドを実行することで Amazon S3 に対して 2 つのパラレルスレッド（`-N2`）で GNU Parallel を実行します。

Aspera Direct-to-S3 の使用

本書で説明するファイル転送プロトコルでは TCP を使用しますが、TCP は経路のレイテンシーが長く、最適とは言えません。このような場合は、UDP を使用すると、処理速度が上がり、パフォーマンスが向上します。

Aspera は UDP をベースとする独自のファイル転送プロトコルを開発し、インターネット上で高速のファイル転送の体験を実現します。Aspera が提供する製品の 1 つである Direct-to-S3 は、UDP ベースのファイル転送プロトコルを提供し、大量のデータを高速で Amazon S3 に直接転送できます。ローカルなデータセンターに大量のデータが格納され、このデータを AWS（例えば Amazon EMR）で後で処理するために Amazon S3 に移動する場合、Aspera Direct-To-S3 は HTTP、FTP、SSH、またはなんらかの TCP をベースとするプロトコルなど他のプロトコルと比べより高速で Amazon S3 にデータを移動できます。

Aspera クラウドベースの製品については、Aspera（http://cloud.asperasoft.com/big-data-cloud/）を参照してください。

AWS Import/Export の使用

AWS Import/Export により、転送用のポータブル記憶装置を用いて AWS 内外へ、大容量データの転送を高速化できます。AWS なら、Amazon の高速内部ネットワークを駆使し、インターネットを使うことなく、ストレージデバイスとの間で直接データを転送できます。大きなデータセットでは、通常 AWS Import/Export のほうがインターネットベースのデータ転送よりも高速であり、通信環境をアップグレードするより経済的にお得です。

AWS Import/Export 使用するには

1. サポートされているデバイスのリストからポータブルストレージデバイスを用意します。詳細については、Selecting Your Storage Device（http://aws.amazon.com/importexport/#supported_devices）を参照してください。

2. Amazon S3 バケット、Amazon Elastic Block Store（EBS）または Amazon Glacier リージョン、AWS アクセスキー ID、および返品お届け先住所を含む Create Job リクエストを AWS に送信します。それに対して、独特なジョブ識別子、お客様のデバイスを認証するためのデジタル署名、およびお客様のストレージデバイスの発送先 AWS 住所が返信されます。

3. 安全にデバイスを識別して認証します。Amazon S3 の場合は、お客様のデバイスのルートディレクトリに署名ファイルを配置します。Amazon EBS または Amazon Glacier の場合は、署名バーコードをデバイスの外側にテープで貼ります。

4. デバイスをそのインターフェイスコネクタおよび電源と一緒に AWS へ発送します。

お客様の荷物が到着すると、処理が行われて、AWS データセンターまで安全に転送されます。そしてここで、AWS Import/Export ステーションに接続されます。データの読み取りが完了すると、AWS がデバイスを返却します。

AWS Import/Export を使用する一般的な方法の 1 つとしては、AWS への初期データ転送および一括データアップロードとしてこのサービスを使用します。このデータのインポートが完了した後、本書で後述するデータの収集および集約フレームワークを使用して、すでにアップロードしたデータにインクリメンタルにデータを追加できます。

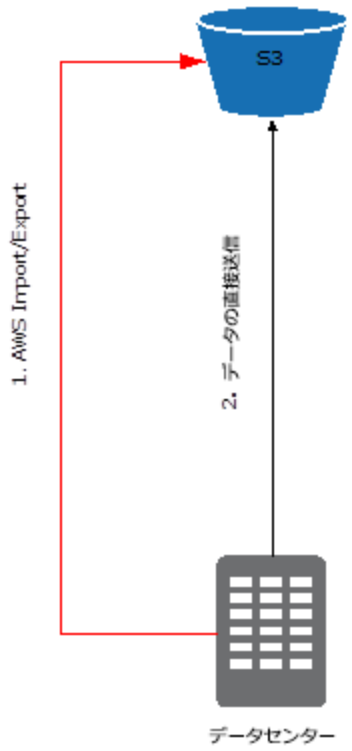

図 3: AWS Import/Export を使用した AWS へのデータの移動

AWS Direct Connect の使用

AWS Direct Connect により、お客様の設備から AWS への専用ネットワーク接続を簡単に確立することができます。AWS Direct Connect を使用すると、AWS とデータセンター、オフィス、またはコロケーション環境間にプライベート接続を確立することができます。これにより、多くの場合、ネットワークのコストを削減する、帯域幅のスループットを向上させる、インターネットベースの接続よりも一貫性のあるネットワークの体験を提供することができます。

AWS Direct Connect では、お客様のネットワークと AWS Direct Connect のいずれかのロケーション間に専用のネットワーク接続を確立できます。業界標準の 802.1q VLAN を使用して、この専用接続を複数の仮想インターフェースに分割することができます。このようにすると、同じ接続を使用して、パブリックリソース（例えば Amazon S3 に格納されたオブジェクト）にはパブリック IP アドレススペースを使用してアクセスし、プライベートリソース（例えば Amazon Virtual Private Cloud（VPC）内で実行されている Amazon EC2 インスタンス）にはプライベート IP スペースを使用してアクセスすることができるので、パブリック環境とプライベート環境の間でネットワークを分離できます。ニーズの変化に合わせて随時仮想インターフェースを再設定できます。

AWS Direct Connect を使用して AWS 上のデータを処理する場合、2 つのアーキテクチャパターンが最も一般的です:

1. **AWS への一括データ転送。** データの大半が AWS に転送されると、Direct Connect ラインを終了し、「データ収集および集約」セクションで説明する方法を使用して、続けてすでに AWS 上に移行したデータの更新を開始できます。この手法では、コストを管理でき、AWS へのデータアップロード中の Direct-Connect リンクの料金だけで済みます。
2. **AWS Direct Connect を使用して、データセンターを AWS リソースと接続します。** 一度接続すると、Amazon EMR を使用して、自社のデータセンターに格納されたデータを処理し、その結果を AWS に格納または再度データセンターに戻すことができます。この手法では、常時 AWS に対して 1 秒間 1 または 10 ギガビットのリンク接続を提供します。また、Direct-Connect のアウトバウンド帯域幅は、インターネット公衆回線のアウトバウンドよりもコストがかかりません。このため、自社のデータセンターに大量のトラフィックがエクスポートされることが予想される場合は、Direct Connect を設定することで、帯域幅の料金を低減できます。

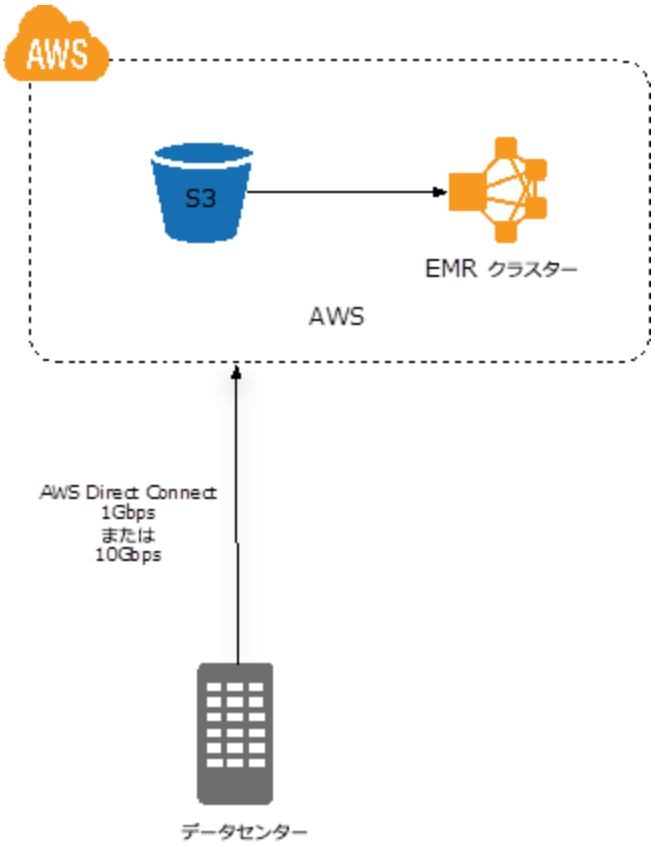

図 4: Amazon Direct Connect を使用した AWS へのデータの移動

シナリオ 3: Amazon S3 から HDFS への大量のデータの移動

AWS（Amazon S3 または Amazon EC2）にデータを移動するだけでなく、Amazon S3 からインスタンス（例えば HDFS）へのデータの移動が必要となる場合があります。このユースケースの詳細については本書で後述しますが、Amazon EC2 にデータを移動するための 2 つの手法を簡単に紹介しましょう。これらの手法では、HDFS へのデータの移動が中心になります。

S3DistCp の使用

前述したように、S3DistCp ではデータセンターの HDFS ストレージから Amazon S3 に大量のデータをコピーできます。しかし、同じツールと類似プロセスを使用して、Amazon S3 からローカルな HDFS にデータを移動することもできます。Amazon EMR を使用し、HDFS にデータをコピーする場合、Amazon EMR コマンドラインインターフェース（CLI）（http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-cli-install.html）を使用して S3DistCp を実行するだけです。以下の例では、S3DistCP での方法を示します:

```
elastic-mapreduce -j JOFLOWID --jar  /home/hadoop/lib/emr-s3distcp-1.0.jar
--step-name "Moving data from S3 to HDFS"
--args "--src,s3://somebucket/somedir/,--dest,hdfs:///someHDFSdir/”
```

注意: *elastic-mapreduce* は Amazon EMR ruby クライアントで、http://aws.amazon.com/developertools/Elastic-MapReduce/2264 からダウンロードできます。上記の例では、`s3://somebucket/somedir/` から `hdfs:///somehdfsdir/` にデータをコピーします。

DistCp の使用

また、DistCp を使用して Amazon S3 から HDFS へデータを移動することもできます。以下のコマンドライン例に、DistCp を使用して Amazon S3 から HDFS にデータを移動する方法を示します:

```
hadoop s3n://awsaccesskey:awssecrectkey@somebucket/mydata/ distcp hdfs:///data/
```

S3DistCP は Amazon S3 との間でのデータの移動に最適化されているため、通常はデータ転送のスループットを向上させるために S3DistCp を使用することをお勧めします。

データ収集

Hadoop で大量のデータを処理する上でのよくある問題の 1 つは、オリジンから最終的な収集ポイントへのデータの移動です。アプリケーションとコンピューティングリソースが短時間で大量のデータを生成するクラウドのコンテキストにおいては、スケーラブルで信頼性のある方法でデータを収集することがビッグデータアーキテクチャで重要な役割を果たします。

従来、アーキテクチャのさまざまな部分で生成されたデータはテキストファイルに格納され、RYSNC、SSH、FTP、その他サードパーティ製ツールなどにより宛先に転送されていました。また、開発者が PHP、Ruby、その他のプログラミング言語で独自のログ収集ツールを作成するのも一般的です。このようにカスタムなデータコレクターを作成することは重要ですが、AWS ユーザーはスケーラブルで効率的な分散データ収集を提供するためにすでに作成されているフレームワークを利用できます。

今日、非常にスケーラブルで信頼性のある方法でソースから宛先にデータを移動できるよう、多様なアプリケーションおよびフレームワークが提供されています。ここで示すツールを使用することで、固有のログ収集およびフレームワークを構築することなく、AWS にデータを容易にロードできます。

Apache Flume の使用

Apache Flume は、大量のデータを効率的に収集、集約、移動するために使用可能な信頼性のある分散型サービスです。ストリーミングデータフローに基づくシンプルでありながら柔軟性のあるアーキテクチャを装備しています。堅牢で耐障害性が高く、調整可能な信頼性メカニズムと多くのフェイルオーバーおよびリカバリメカニズムを備えています。

Apache Flume には、エージェント、コレクター、マスターといった概念があります。エージェントは、ウェブログ、API ログ、syslog、その他定期的に発行されたデータなどを収集します。Flume エージェントをデータソース（ウェブサーバー、ad サーバー、またはその他データを生成するソース）にインストールし、コレクターにデータを発送できます。コレクターの役割は、エージェントからデータを収集して、Amazon S3 や HDFS などの永続的なストレージに格納することです。

Flume の主な利点の 1 つは、そのシンプルでありながら強力な階層データ収集モデルであり、これによりエージェントノードを増やし、コレクターを減らしてデータを収集するセットアップが可能です。

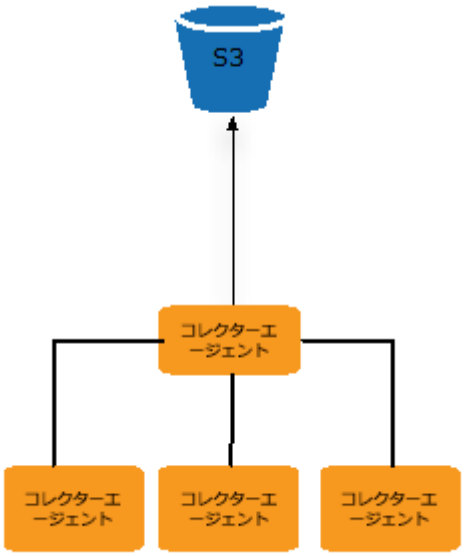

図 5: Apache Flume を使用したデータ収集

Fluentd の使用

Fluentd（Treasure Data 提供）のアーキテクチャは Apache Flume または Facebook の Scribe と非常によく似ています。Fluentd は Flume や Scribe よりもインストールおよび管理が簡単で、適切なドキュメントおよびサポートが提供されています。

Fluentd は 3 つの基本コンポーネントから構成されます:

- **Input** は、Fluentd が入力としてログを読み取る場所を示します。Input は拡張可能で、さまざまなソースからのログのアップロード形式をサポートできます。正式にサポートされている入力形式は HTML、JSON、テーリングファイル、syslog です。
- **Buffer** は、入力メモリバッファを示します。出力ステップが失敗した場合、Fluentd はメモリにログをバッファします。
- **Output** は、出力データの格納位置を示します。Fluentd がバッファから Output にデータを渡すと、Output は永続的ストレージに書き込みます。コミュニティでは、Amazon S3、Amazon SQS、MongoDB、Redis などの出力をサポートしています。

Flume と同じように、Fluentd は階層型の収集アーキテクチャをサポートし、Fluentd ノードセットがデータを収集し、集約およびデータ保持のために別の Fluentd ノードセットに転送します。

データ集約

データ集約は、個々のデータレコード（ログレコードなど）を収集し、大きなデータファイルにまとめるための手法を示します。例えば、ウェブサーバーログ集約では、単一のログファイルで最新のすべての訪問を記録します。

Amazon EMR でのデータレコード集約には、いくつかの利点があります:

- AWS にデータをアップロードするために必要な回数を減らすことで、データ取り込みのスケーラビリティが向上します。つまり、小さなファイルをたくさんアップロードするのではなく、少数の大きなファイルをアップロードします。
- Amazon S3（または HDFS）に格納されるファイルの数が減り、本質的に Amazon EMR 上のデータの処理時のパフォーマンスが向上します。Amazon EMR が動作する Hadoop は、小さいファイルが多い場合と比べ、大きいファイルが少ない方が通常パフォーマンスが向上します。
- 圧縮比率が高くなります。圧縮可能な大きなファイルを圧縮したほうが、小さいファイルをたくさん圧縮するよりも通常は効果的です。

分散ログ収集と同様、最終的な宛先（Amazon S3 または HDFS）にコピーする前にデータを集約するように、Flume や Fluentd のようなログコレクターを設定できます。集約をサポートしていないフレームワークを使用し、最終的な格納場所に小さなファイルが多くある場合、S3DistCp の集約機能を利用できます。http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/UsingEMR_s3distcp.html に提供される Amazon EMR ガイドの S3DistCp セクションを参照してください。

Apache Flume でのデータ集約

Flume のコレクター設定で、データを格納する前に、入力データやログを集約するように Flume に指示できます。5 分ごとにデータファイルを集約する設定ファイル（`/etc/flume/conf/flume-site.xml`）の例を示します:

```
<configuration>
  <property>
    <name>flume.master.servers</name>
    <value>ec2-master </value>
  </property>
  <property>
    <name>flume.collector.output.format</name>
    <value>raw</value>
  </property>
  <property>
    <name>flume.collector.dfs.compress.codec</name>
    <value>GzipCodec</value>
  </property>
  <property>
    <name>flume.collector.roll.millis</name>
    <value>300000</value>
  </property>
</configuration>
```

データ集約 ベストプラクティス

データ集約の推奨ベストプラクティスを理解するには、Hadoop でのデータ処理を理解すると役立ちます。

データを処理する前に、Hadoop はデータ（ファイル）を複数のチャンクに分割します。ファイルの分割後、1 つのマップタスクがそれぞれのパートを処理します。HDFS を基盤のデータストレージとして使用している場合は、HDFS フレームワークがすでにデータファイルを複数のブロックに分割しています。さらに、データがフラグメント化されるため、Hadoop は HDFS データブロックを使用して、単一のマップタスクを各 HDFS ブロックに割り当てます。

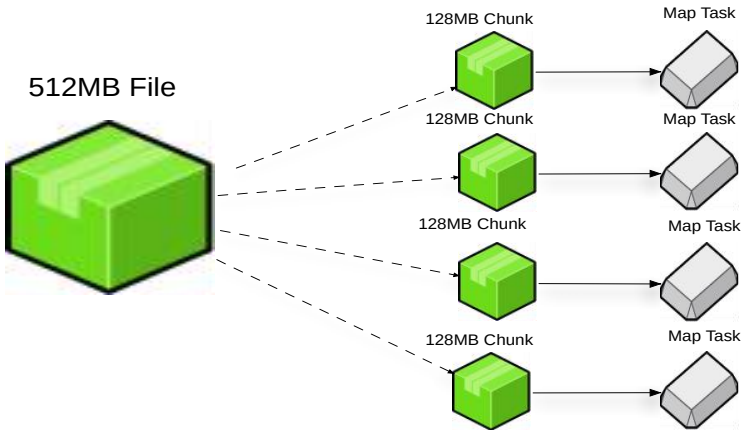

図 6: Hadoop の分割ロジック

Amazon S3 に格納されているデータにも同じ分割ロジックが適用されるものの、そのプロセスは異なります。Amazon S3 上のデータは HDFS 上のデータと同じようには複数の部分に分割されないため、Hadoop は複数の HTTP レンジリクエスト内のファイルを読み込むことで、Amazon S3 上のデータを分割します。簡単に言えば、HTTP はファイル全体ではなくファイルの一部をリクエストします（例えば、`GET FILE X Range: byte=0-10000`）。Hadoop が Amazon S3 からデータファイルを読み取るために使用する分割サイズは、使用している Amazon EMR Amazon マシンイメージ（AMI）バージョンによって異なります。最新バージョンの Amazon EMR では従来よりも分割サイズが大きくなっています。例えば、Amazon S3 上の単一のデータファイルがおよそ 1 GB の場合、Amazon S3 の分割サイズが 64 MB とすると、15 種類の HTTP リクエスト（1 GB/64 MB = ~15）を並列で発行することにより、Hadoop は Amazon S3 からファイルを読み取ります。

圧縮サイズのベストプラクティスを実装するには、圧縮アルゴリズムが Hadoop のデータ分割ロジックに与える影響を理解しておくと有益です。データの格納場所（HDFS または Amazon S3）に関係なく、圧縮アルゴリズムが分割に対応していない場合、[5]Hadoop はデータファイルを分割せず、単一のマップタスクを使用して圧縮ファイルを処理します。例えば、データファイルが 1 GB の GZIP ファイルの場合、Hadoop は単一のマッパーでファイルを処理します。一方、ファイルを分割できる場合（テキストファイルのケース、または LZO の一部のバージョンなど分割が可能な圧縮のケース）は、Hadoop はファイルを複数のチャンクに分割して、そのチャンクを並列処理します。

[5] 本書で後述する「どの**圧縮アルゴリズムを使用すべきか?**」を**参照してください。**

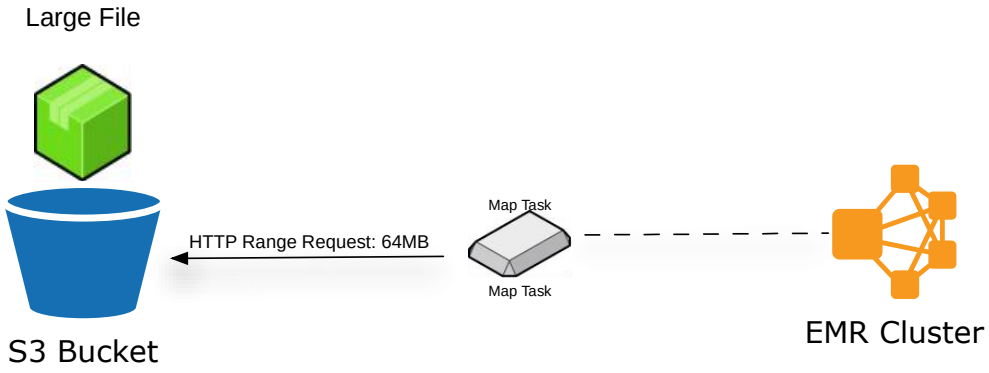

図 7: Amazon EMR の圧縮データの S3 からの取得

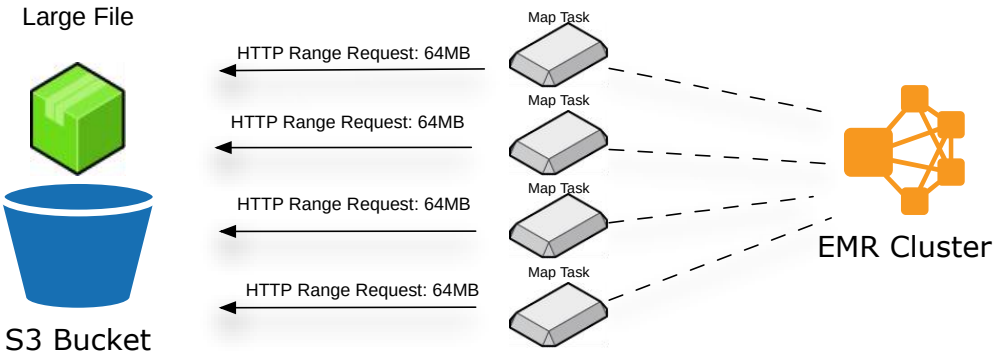

図 8: Amazon EMR の HTTP レンジリクエストの使用

ベストプラクティス 1: 集約されるデータのサイズ

集約される適切なデータファイルのサイズは、使用している圧縮アルゴリズムによって決まります（データ圧縮については、本ペーパーで後述します）。例えば、ログファイルが GZIP で圧縮されている場合、通常、集約されるデータファイルのサイズを 1～2 GB にしておくことが最適です。この理由は、GZIP ファイルは分割できないため、Hadoop はデータを処理するために単一マッパーを割り当てるからです。つまり、単一マッパー（単一スレッド）が Amazon S3 からデータを取得します。単一スレッドはその時点で Amazon S3 から取得できるデータの量（スループット）に制限されるため、Amazon S3 からマッパーにファイル全体を読み込むプロセスはデータ処理ワークフローでボトルネックになります。一方、データファイルを分割できる場合は、複数の単一マッパーでファイルを処理できます。このようなデータファイルに適切なサイズは 2～4 GB です。

ベストプラクティス 2: データ集約サイズの制御

重要な問いは、データ集約サイズをどのように制御するかということです。上記のセクションで説明した分散ログコレクターを使用している場合、時間に基づくデータ集約に制限されます。例えば、集約されたファイルを Amazon S3 に送信する前に 5 分間データを集約するように Flume を設定できます。残念ながら、時間集約では、作成されるデータファイルの正確なサイズを自由に制御することはできず、ファイルのサイズは分散ロジックコレクターのデータ読み取り速度によって決まります。例えば、Flume が 1 MB/秒の速度でデータを読み取る場合、5 分ごとに集約すると、ファイルは最大 300 MB になる可能性があります。

本書の執筆時点では、ファイルサイズ別で集約できるログ収集フレームワークはなく、そのため、適正な時間ベースの集約を見つけ出すことは多くの場合、試行錯誤の連続です。ファイルサイズの集約ロジックがない場合は、集約設定で異なる時間値を各自テストして、どの値が最善のファイルサイズ集約であるか判断することをお勧めします。例えば、5 分ごとにログファイルを集約するように Fluentd を設定し、テストした結果、5 分の集約により 4 GB の GZIP で圧縮されたファイルが作成されることが判明したとします。直前のセクションを思い出すと、2 GB よりも大きいファイルを Amazon S3 に格納して Amazon EMR で処理した場合、ログジャムが発生します。この場合、集約の時間値を 2 分以下に下げることができます。

全体的に見れば、ほとんどの分散ログコレクターフレームワークはオープンソースであるため、選択したログコレクター専用のプラグインを作成し、ファイルサイズに基づく集約機能を導入することができます。

ベストプラクティス 3: データ圧縮アルゴリズム

集約されたデータファイルの大きさに応じ、圧縮アルゴリズムは重要な選択になります。例えば、データを集約し（好みの取り込みツールを使用）、集約されたデータファイルが 500 MB～1 GB の場合、GZIP 圧縮が許容されるデータ圧縮タイプです。ただし、データ集約で 1 GB よりも大きいファイルが作成される場合は、分割をサポートしている圧縮アルゴリズムを選択することが最適です。

データ圧縮

データの圧縮は、いくつかの理由で重要です:

1. データストレージのフットプリントを小さくすることで、ストレージコストを削減します。
2. ソースから宛先に移動させるデータを少なくすることで、帯域幅コストを低減します。
3. データの格納場所、マッパー、リデューサーの間で移動させるデータを少なくすることで、データ処理のパフォーマンスが向上します。
4. Amazon EMR がディスクに書き込むデータを圧縮することにより、データ処理のパフォーマンスが向上します。つまり、ディスクに書き込む頻度が少なくなることで、パフォーマンスの向上を実現します。

どの圧縮アルゴリズムを使用すべきか?

当然ながら、圧縮アルゴリズムはすべて同じではありません。以下のような潜在的な利点と欠点を考慮してください:

- 下表に示すように、高速な圧縮アルゴリズムがあります。高速な圧縮が役立つかどうか判断するには、ワークロードを理解する必要があります。例えば、CPU 制約を受けるジョブの場合、高速な圧縮アルゴリズムでは、パフォーマンスが十分向上しない可能性があります。圧縮スピードが重要であると判断した場合は、Snappy 圧縮が高速であると思われます。
- 圧縮アルゴリズムによっては、速度は遅いものの、より多くのスペースを節約できるものもあり、場合によってはこうしたアルゴリズムのほうが重要かもしれません。ただし、ストレージコストが重要な要因でない場合は、高速なアルゴリズムが適しているかもしれません。
- 重要なことに、ファイル出力を分割できるアルゴリズムもあります。前述したように、データファイルを分割する場合、データファイルの格納方法に影響が生じます。圧縮アルゴリズムが分割をサポートしていない場合は、ファイルサイズを小さく抑える必要があります。ただし、圧縮ファイルをチャンクできる場合は、大きなファイルを格納して、Amazon EMR で処理することができます。

圧縮	拡張子	分割可能	圧縮/解凍速度（スケール 1～4）	圧縮率 %（スケール 1～4）
Gzip	gz	いいえ	1	4
LZO	lzo	はい（インデックス化されている場合）	2	2
Bzip2	bz2	はい	3	3
Snappy	snappy	いいえ	4	1

図 9: 圧縮形式の比較

マッパーおよびリデューサーの出力の圧縮

Amazon EMR データ処理では、入力ファイルの圧縮だけが効果的であるわけではありません。マッパーからの中間出力の圧縮もデータ処理ワークフローの最適化につながります。これは、マッパーがデータの出力を完了した後、Amazon EMR がリデューサーにコピーするデータです。圧縮により、ネットワーク上にコピーするデータの量を減らすことができます。ただし、中間データを圧縮することによるパフォーマンスの向上は、リデューサーにコピーしなければならないデータの量に完全に依存します（注意: Hadoop がネットワークを介してコピーするデータの量を調べる 1 つの方法として、Hadoop の `REDUCE_SHUFFLE` を確認できます）。マッパーの中間データの圧縮を有効にするには、`mapreduce.map.output.compress` を true に設定し、`mapreduce.map.output.compress.codec` を好きな圧縮コーデック（GZIP、LZO、または Snappy）に設定します。

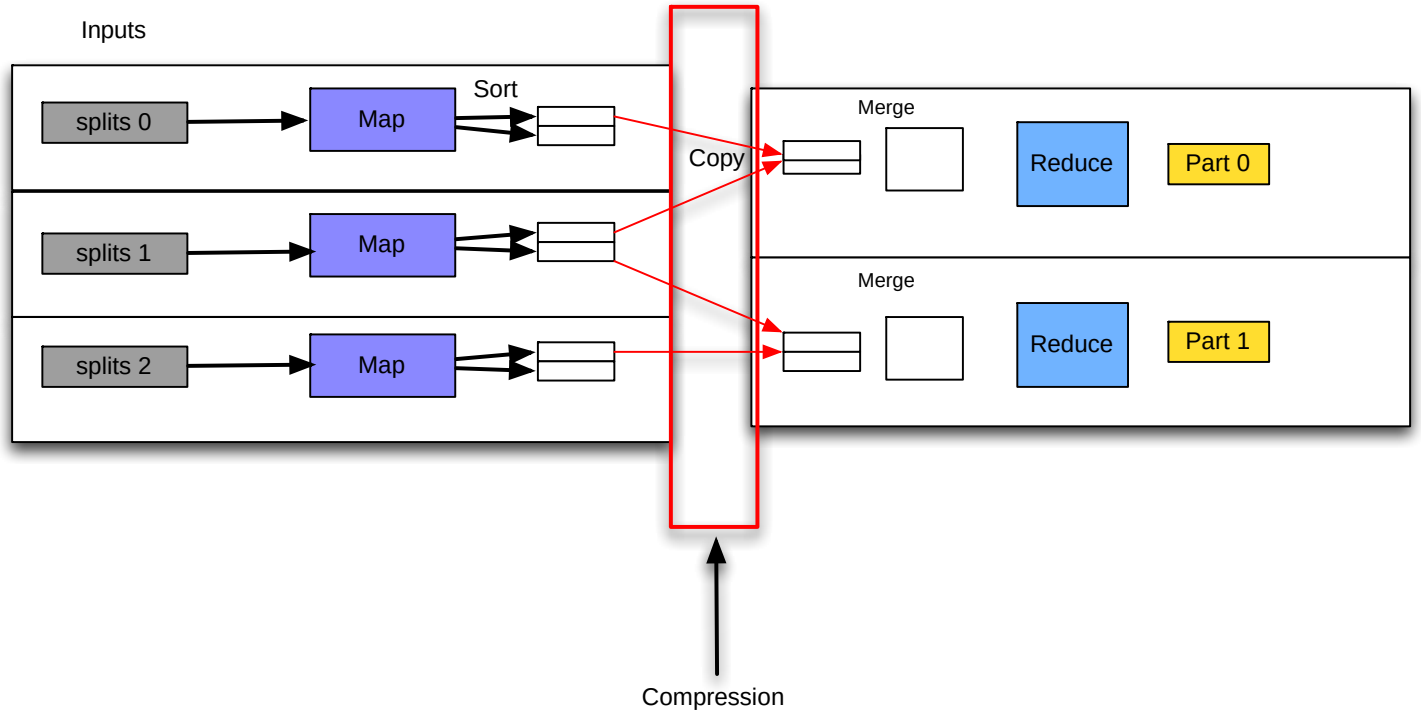

図 10: マッパー出力の圧縮

もう 1 つデータ圧縮を使用できるのは、マッパーからディスクへデータを書き出す、またはデータを書き込むときです。マッパーがデータを処理する場合、マッパーの出力がメモリバッファに格納されます。マッパーのバッファのサイズは制限されていて（設定可能）、マッパーのバッファ容量よりも入力データが大きいと、バッファ内部のデータはディスクに書き出されます。圧縮をオンにすると、ディスクに書き込むデータ量が少なくなります。LZO は、マッパーの出力データを圧縮するために適した圧縮候補の 1 つです。圧縮を有効にするには、Hadoop ジョブでパラメータ `mapred.compress.map.output=true` が設定されていることを確認してください。

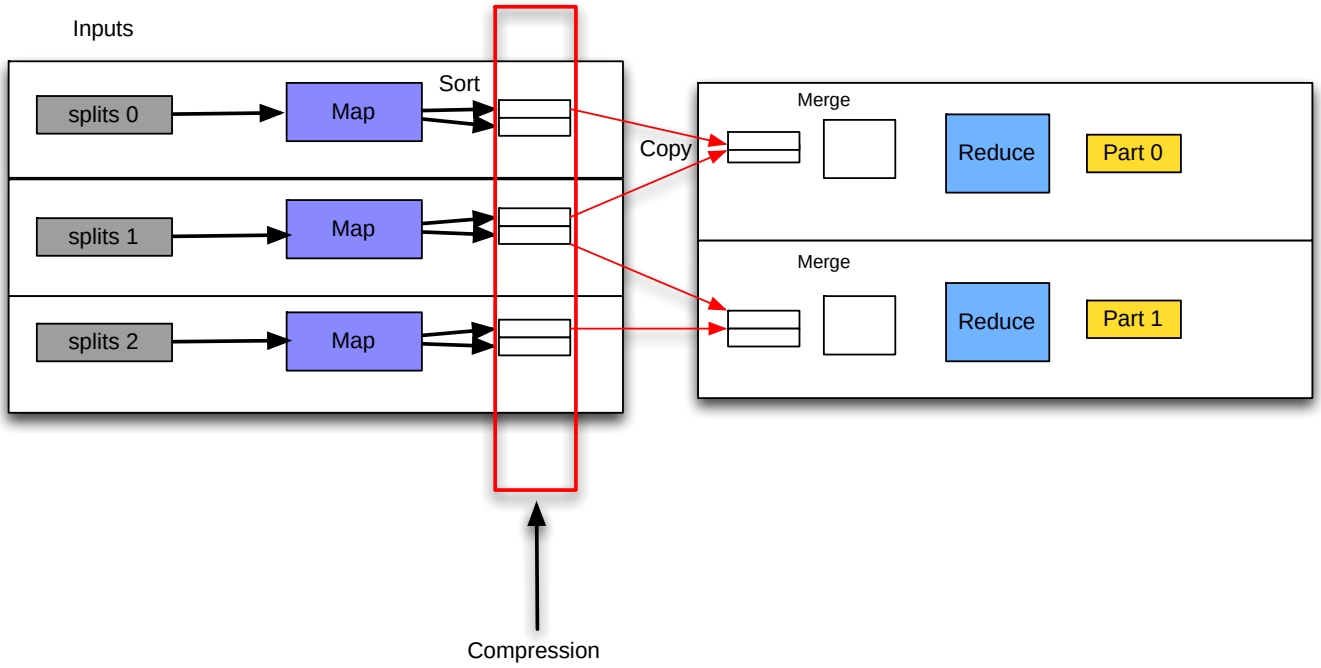

図 11: マッパーの中間書き出しの圧縮

要約すると、ワークロードに推奨される圧縮アルゴリズムは以下の要因によって決まります:

- データファイルをどれだけ速く圧縮および解凍する必要があるか。
- どの程度データストレージスペースを節約したいか。
- データファイルが小さく処理に複数のマッパー（データファイルの分割）を必要としないのか、それともファイルが大きく同時に処理するには複数のマッパーが必要なのか（この場合、分割をサポートする圧縮アルゴリズムが必要）。
- 可能であれば、GZip など、Java を実装した場合よりも高いパフォーマンスを実現できるネイティブな圧縮ライブラリを選択してください。

ベストプラクティス 4: データのパーティション

データのパーティションは、データ処理ワークフローに不可欠な最適化です。データのパーティションを設定していないデータ処理ジョブでは、提供されているすべてのデータセットを読み取りまたはスキャンし、不要なデータをスキップするためにさらにフィルタを適用する必要があります。このようなアーキテクチャはデータ量が少ない場合は有効かもしれませんが、大きなデータセットではデータセット全体をスキャンするのは時

間も費用もかかります。データのパーティションにより、固有のデータバケットを作成でき、データ処理ジョブでデータセット全体を読み取る手間を解消できます。

データをどのようにパーティションするかは、以下の 3 つを考慮して決定します。

- データ型（時系列）
- データ処理頻度（1 時間おき、1 日おきなど）
- データアクセスおよびクエリパターン（時間のクエリ vs. 地理的場所のクエリ）

例えば、時系列のデータセットを処理し、1 時間おきにデータを処理する必要があり、データアクセスパターンが時間に基づく場合は、日時に基づいたデータのパーティションが最も効果的です。このようなデータ処理の一例が毎日のログの処理です。さまざまなデータソース（ウェブサーバー、デバイスなど）からログを受信する場合、時刻別にデータのパーティションを作成すると、日時に基づいたパーティション体系になります。このようなパーティショニング方式は以下のような構造になります:

`/data/logs/YYYY-MM-DD-HH/logfiles` この時刻に対し（ここで、YYYY-MM-DD-HH は最新のログ取得時間に基づいて変動）:

```
/data/logs/2013-01-01-02/logfile1
                        .../logfile2
                        .../logfile3

/data/logs/2013-01-01-03/logfile1
                        .../logfile2
                        .../logfile3
```

Amazon EMR によるデータの処理

Amazon EMR によりデータ処理は非常に簡単かつ便利になりますが、いくつかの点を考慮することによりデータ処理ワークフローはさらに効果的になります。

適正なインスタンスサイズの選択

Amazon EMR クラスターをプロビジョニングする場合、ワークロードによって CPU を大量に使用するものもあれば、ディスク I/O またはメモリを大量に使用するものもあるので、ノードのインスタンスサイズは重要です。

メモリを大量に使用するジョブでは、m1.xlarge またはいずれかの m2 ファミリのインスタンスサイズであれば、処理に十分なメモリおよび CPU 能力があります。CPU を大量に使用するジョブの場合は、c1.xlarge、cc1.4xlarge、または cc2.8xlarge インスタンスが最適です。通常はメモリと CPU の両方を大量に使用するジョブであり、このようなワークロードを処理するのに適したメモリおよび CPU 能力を備えた cc1.4xlarge または cc2.8xlarge インスタンスサイズを選択するのが最適です。

本書の圧縮セクションで、Hadoop はできる限り多くのメモリを使用しようとするということを学習しましたが、データを処理するのに十分なメモリがない場合（大量のデータで並べ替えやグループ化を行う場合など）は、データの一部がディスクに書き出されます（書き込まれます）。ディスク書き込みは高コストでジョブの処理速度が低下するため、高いパフォーマンスを実現するためにメモリが多いインスタンスを使用します。

Amazon EMR の主要な利点の 1 つが、異なるインスタンスサイズを試すことができることであることに留意してください。クラスターをシャットダウンし、要件に適した別のインスタンスサイズに変えて新しいクラスターを実行するだけで、異なるインスタンスプロファイル（高メモリと高 CPU）に切り替えることができます。

本書の後半で、Amazon EC2 インスタンスの異なるタイプを混在させ、CPU とメモリ両方を大量に使用するジョブの需要に合わせてクラスターサイズを拡大または縮小できるアーキテクチャパターンについて学習します。Amazon EMR CLI またはコンソールを介してクラスターをプロビジョニングする場合、Amazon EMR はインスタンスサイズに応じて、以下のような Hadoop 固有の適切な構成で各インスタンスを構成します:

- デーモンあたりの Java メモリ（ヒープ）のサイズ
- インスタンスあたりのマッパーの数
- インスタンスあたりのリデューサーの数

Hadoop 構成は Amazon EMR クラスターをブートストラップすることにより変更できますが（後述）、Amazon EMR チームがテストし、微調整済みである事前設定パラメータをそのまま使用することをお勧めします。Hadoop 構成の変更は、Hadoop の構成に精通されている方以外にはお推めできません。

ジョブに適正なインスタンスの数の選択

クラスターに適正なインスタンスの数は、データセットのサイズおよび Hadoop のデータ処理速度によって決まります。Hadoop は並列処理できるようファイルをチャンクに分割して入力データファイルを処理することを思い出してください。データセットが大きいほど、Hadoop が作成する分割が増え、Hadoop はデータを処理するためにより多くのタスクを実行することになります。Amazon EMR クラスターの適正なサイズは、十分なノードを持ち、多くのマッパーおよびリデューサーを並列処理できる大きさです。例えば、入力ファイル（Hadoop により 20 に分割）を処理するために 20 のマッパーを必要とするデータセットの場合、最適な Amazon EMR クラスターであれば、すべてのマッパーを並列処理できます。この場合、最適な Amazon EMR クラスターを実行するには、すべてのマップタスクジョブを並列で実行するのに 10 の m1.small インスタンスが必要です（1 つの m1.small で 2 つのマップタスクを並列で実行できます）。

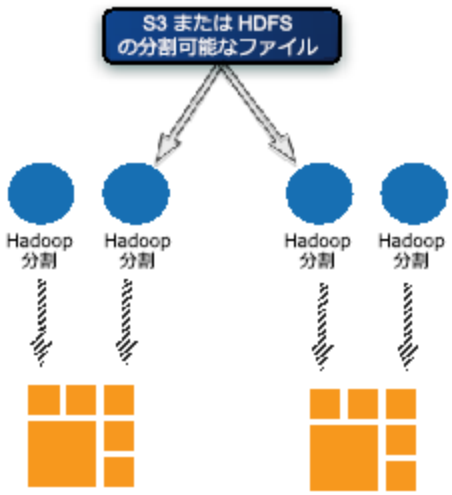

図 12: Hadoop の並列分割処理

ただし、以下のケースでは、すべてのマッパーを並列で実行する必要がありません:

- データセットが大きく、コスト面で、マッパータスクを並列で実行できる Amazon EMR クラスターを作成できない。
- 時間の制約がない。

上記のような状況では、Amazon EMR クラスターは少ないノード数で高いコスト効果を達成できます。すべてのマッパーを並列で実行できるだけのノードやマッパー容量がない場合、Hadoop は残りのマッパーをキューに登録し、使用可能な容量を確保できた時点で、残りのマッパーを処理します。このようにしてキューの処理を続け、下図に示すように、すべてのマッパーを実行します。

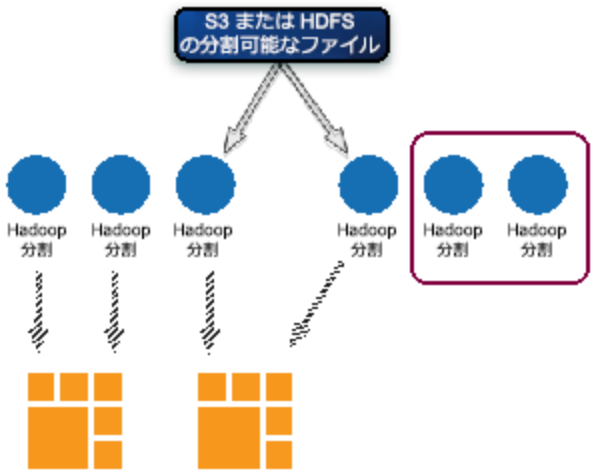

図 13: Hadoop のキューの分割

上記の例（データを処理するのに Hadoop で 20 のマッパーを必要とする入力サイズの場合）を再度使用し、今回は m1.small インスタンスを 5 つにしてみます。m1.small Amazon EMR ノードが 5 つの場合、10 のマッパーは並列処理できますが、残りの 10 のマッパーはキューに登録されます。Hadoop が最初の 10 のマッパーを処理してから、残りの 10 のマッパーが動作します。すべてのマッパーが並列で動作するわけではないため、ノードの数を減らすことでデータの処理時間が向上します。

要約すると、Amazon EMR インスタンスの適正な数は、データをどれだけ迅速に処理する必要があり、データの処理時間要件にどれだけの金額を支払うかによって決まります。Amazon EMR の料金は 1 時間単位で発生します。つまり、ジョブの実行に 1 時間かからなかった場合でも、1 時間分の EC2 利用料金が請求されます。したがって、コストを削減するには、Amazon EMR インスタンスの数を減らして、処理時間を短縮することをお勧めします。

クラスターが並列で実行できるマッパーの数は、Amazon EMR クラスターインスタンスのサイズとインスタンスの数によって決まります。インスタンスタイプごとのマッパーの数は、開発者ガイド（http://docs.amazonwebservices.com/ElasticMapReduce/latest/DeveloperGuide/TaskConfiguration_AMI2.2.html）を参照してください。例えば、10 の m1.large インスタンスから構成されるクラスターの場合、30 のマッパーを並列で実行できます。

ジョブで必要なマッパーの数の予測

入力データファイルを処理するために必要なマッパーの数を予測するには、少なくとも 2 つの方法があります:

1. マッパーの数は Hadoop の分割の数によって決まります。HDFS または Amazon S3 の分割サイズよりもファイルが小さい場合、マッパーの数はファイル数と同じになります。ファイルの一部または全部が HDFS または Amazon S3 の分割サイズ（fs.s3.block.size）よりも大きい場合、マッパーの数は HDFS/Amazon S3 ブロックサイズで割った各ファイルの合計と等しくなります。

   以下の例では、64 MB のブロックサイズ（S3 または HDFS）を想定します。

   例 1: それぞれ 60 MB のファイルが HDFS に 100 ファイルある = 100 マッパー。各ファイルがブロックサイズよりも小さいので、マッパーの数はファイルの数と同じです。

   例 2: それぞれ 80 MB のファイルが Amazon S3 に 100 ファイルある = 200 マッパー。各データファイルはブロックサイズよりも大きく、このため各ファイルではそのファイルを処理するためにマッパーが 2 つ必要になります。100 ファイル * 各 2 マッパー = 200 マッパー

   例 3: 60 MB が 2 ファイル、120 MB が 1 ファイル、10 MB が 2 ファイル = 6 マッパー。60 MB ファイルではマッパーが 2 つ、120 MB ファイルではマッパーが 2 つ、2 つの 10 MB ファイルではそれぞれマッパーが 1 つずつ必要です。

2. 必要なマッパーの数を予測するための簡単な方法は、いずれかの Amazon EMR クラスター内のジョブを実行し、ジョブに対して Hadoop が算出したマッパー数を記録することです。JobTracker GUI またはジョブの出力を調べることで、この合計が判明します。ジョブ出力のサンプルを以下に示します。マッパー数は強調表示されます:

```
13/01/13 01:12:30 INFO mapred.JobClient:   Total time spent by all reduces waiting after reserving slots (ms)=0
13/01/13 01:12:30 INFO mapred.JobClient:   Total time spent by all maps waiting after reserving slots (ms) =0
13/01/13 01:12:30 INFO mapred.JobClient:   Rack-local map tasks=20
13/01/13 01:12:30 INFO mapred.JobClient:   Launched map tasks=20
13/01/13 01:12:30 INFO mapred.JobClient:   SLOTS_MILLIS_REDUCES=2329458
```

Amazon EMR のクラスタータイプ

Amazon EMR には、一時的なものと永続的なものの 2 つのクラスタータイプがあります。タスクおよびシステム構成に応じて、いずれかを使い分けることができます。

一時的な Amazon EMR クラスター

一時的なクラスターとは、ジョブまたはステップ（一連のジョブ）が完了するとシャットダウンするクラスターです。これに対して、データ処理が完了しても、永続的なクラスターは実行を継続します。大半の時間クラスターがアイドル状態であることが判明している場合は、一時的なクラスターを使用するのが最善です。例えば、Amazon S3 からウェブログを取得し、1 日に一度データを処理するバッチ処理ジョブの場合、一時的なクラスターを使用してデータを処理し、処理の完了後にノードをシャットダウンするとより経済的です。要約すると、一時的なクラスターは以下の 1 つまたは複数の状況で検討してください。

1. 1 日あたりの Amazon EMR 処理時間の合計が 24 時間未満で、使用していないときにクラスターをシャットダウンすることで利益を得ることができる。
2. プライマリデータストレージとして HDFS を使用していない。
3. データの処理量が多く、反復している。

一時的な Amazon EMR クラスターを使用する必要があるかどうか判断するには、以下の計算式が有効です:

注意: 以下の計算式はあくまでも大まかな目安にすぎません。環境によって変わる可能性があります。

> *1* 日あたりのジョブ数 *（クラスターをセットアップするための時間（*Amazon S3* を使用している場合は *Amazon S3* データのロード時間も含む） + データの処理時間） < *24* 時間の場合は、一時的な *Amazon EMR* クラスターを検討してください。

例えば、Amazon S3 にログファイルを格納し、1 日あたり 10 個のウェブログ処理ジョブを実行し、それぞれのジョブがおよそ 40 分かかる場合、1 日あたりの合計処理時間は 10 時間未満です。このため、ログ処理が完了するたびに、Amazon EMR クラスターをシャットダウンしてください。

上記の計算は、反復的でないデータ処理のほうが有効です。何度も同じデータセットを処理する反復的なデータ処理（例えば、機械学習の場合など）を行っている場合は、HDFS にデータを移動し、代わりに以下の計算式を使用してください:

> （*Amazon S3* から *HDFS* にデータをロードするための時間 + *1* 日あたりのジョブ数） * データ処理時間 < *24* 時間の場合に、一時的な *Amazon EMR* クラスターを検討してください。

永続的な Amazon EMR クラスター

その名前が示すように、永続的な Amazon EMR クラスターはデータ処理ジョブが完了しても実行を継続します。一時的な Amazon EMR クラスターと同様、永続的なクラスターも固有のコストおよび利点があります。永続的なクラスターは、以下の 1 つまたは複数の状況で検討してください:

- 処理ジョブを頻繁に実行し、1 つジョブが完了してもクラスターを実行したままにしておいたほうが有益である。
- 相互の処理ジョブに入出力の依存関係がある。2 つの独立した Amazon EMR クラスター間でデータを共有できるものの、次のジョブを実行するためには前のジョブの結果を HDFS に格納しておくと便利な場合があります。
- まれなケースですが、Amazon S3 ではなく HDFS にデータを格納したほうが経済的なことがあります。

一時的なクラスターと同じように、以下の計算式を使用することで、一時的なクラスターと永続的なクラスターのどちらを使用する必要があるか判断できます:

注意: 以下の計算式はあくまでも大まかな目安にすぎません。環境によって変わる可能性があります。

> *1* 日あたりのジョブの数 *（クラスターをセットアップするための時間（*Amazon S3* を使用している場合は *Amazon S3* データのロード時間も含む） + データの処理時間） > *24* 時間の場合、永続的な *Amazon EMR* クラスターを使用します。

または

> *Amazon S3* から *HDFS* にデータをロードするための時間 + *1* 日あたりのジョブ数 * データ処理時間 > *24* 時間の場合、永続的な *Amazon EMR* クラスターを使用します。

一般的な Amazon EMR のアーキテクチャ

Amazon EMR はさまざまなアーキテクチャ構成で使用でき、それぞれに利点と欠点があります。

パターン 1: HDFS の代わりに Amazon S3 を使用

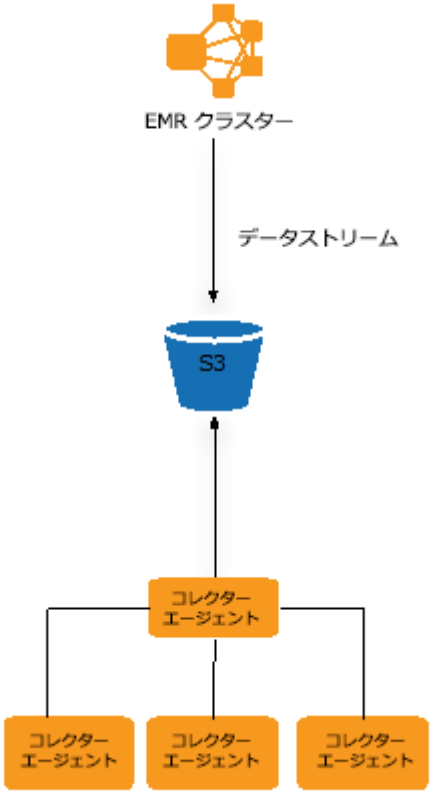

図 14: HDFS の代わりに S3 を使用

このアーキテクチャパターンでは、前述したデータのパーティション化、データサイズ、圧縮のベストプラクティスに従い、Amazon S3 にデータを格納します。データ処理については、Amazon EMR ノードが Amazon S3 からデータを取得し、データをダウンロードしながら処理します。このアーキテクチャでは、Amazon EMR はローカルディスクにデータをコピーするのではなく、マッパーが Amazon S3 へのマルチスレッドの HTTP 接続を開き、データを取得して、ストリーム単位でデータを処理することを理解しておくことが重要です。

このアーキテクチャには多くの利点があります:

- Amazon S3 は耐久性が高いデータストレージを提供します。
- データは Amazon S3 にあり、HDFS にはないので、NameNode の損失に関する心配は限られています。
- セキュリティやライフサイクル管理など、Amazon S3 の柔軟で素晴らしい特徴を利用できます。
- Amazon S3 はほぼ無限にスケーリングします。一般的には、Amazon EMR ノードに対してデータを並列で取得するために必要な数の Amazon EMR ノードを実行できます。HDFS ストレージをスケーリングすることなく、データ処理時間を短縮できます。通常、HDFS 容量を気にする必要はありません。Amazon S3 は非常にスケーラブルなストレージスペース量を提供します。

- Amazon S3 はコスト効果に優れています。Amazon S3 に格納するデータの料金を支払うだけです。これに対して、100 GB のデータを HDFS に格納する場合は、100 GB の格納料金にレプリケーション係数を掛けた額になります。例えば、HDFS に 100 GB のデータが格納され、レプリケーション係数が 2（異なる HDFS ノードにデータが 2 度レプリケーションされる）であると、実際に HDFS に格納されるデータは 200 GB です。大きなデータセットでは、これはより多くの HDFS ストレージでより多くのインスタンスをプロビジョニングする必要があり、Amazon EMR のコストが増大する可能性があることを意味します。
- このアーキテクチャでは、一時的な Amazon EMR クラスターを使用できます。
- HDFS ノードをオーバーロードすることなく、同じデータセットで複数の Amazon EMR クラスター内の複数のジョブを実行できます。
- データの破損に対する保護を強化できます。HDFS では、HDFS データのレプリケーションにより、データ格納後の破損からデータを保護することができます。しかし、HDFS はデータを格納する前に発生するデータの破損は防止できません。これに対して、Amazon S3 では、データセットで Amazon S3 バージョニングを有効にした場合、データの破損を防止できます。
- Amazon S3 はライフサイクル管理を備え、Amazon S3 に格納されているパターンや期間に基づき、データを消去および削除できます。これは、ストレージコストを抑制するための大きな機能です。

このアーキテクチャを使用する場合、少なくとも 1 つの潜在的な欠点があります:

- 複数のパスで何度もデータを処理する必要がある反復的なデータ処理では、これは効率的なアーキテクチャではありません。ネットワークを介して何度も Amazon S3 からデータが取得されるためです。反復ワークロードでは、アーキテクチャパターン #2 Amazon S3 と HDFS を使用することを推奨します。

パターン 2: Amazon S3 と HDFS

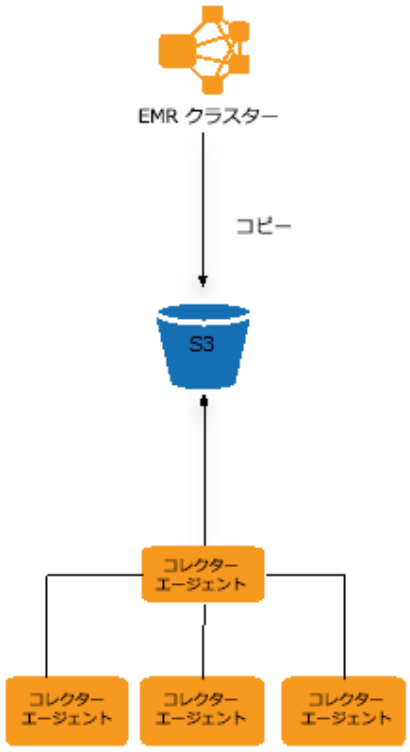

図 15: S3 と HDFS の使用

このアーキテクチャパターンでは、前述したデータのパーティション化、データサイズ、圧縮のベストプラクティスに従い、Amazon S3 にデータを格納します。データ処理ジョブを実行する前に、Amazon EMR は DistCp または S3DistCp を使用して HDFS にデータをコピーします。前記のパターンと比べ、このパターンでは、同じデータセットで反復処理を行う場合に、データを処理するたびに Amazon S3 から Amazon EMR ノードにデータをコピーする必要がないという利点があります。このため、データをローカルディスクにコピーすることにより、反復ワークロードの処理時間を短縮できます。

このアークテクチャには多くの利点があります:

- Amazon S3 は耐久性が高いデータストレージを提供します。
- 以下のような Amazon S3 の素晴らしい柔軟な機能を利用できます:
  - **バケット権限およびアクセスコントロール**により、だれがデータにアクセスでき、どのようなアクションを行うことができるのかを制御できます。
  - **保管データ暗号化**により、サーバー側の暗号化を実現し、保管時にデータを暗号化できます。
  - **ライフサイクル管理**により、格納パターンや期間に基づいてデータを消去または削除できるので、ストレージコストを抑制できます。
- Amazon S3 の大規模なスケールにより、S3DistCp または DistCp を使用してデータを並列で取得するために必要な数の Amazon EMR ノードを実行できます。
- データが HDFS に格納されている場合、データの増大に伴い、容量（HDFS ノード）を追加する必要があります。Amazon S3 はストレージ容量が大きいため、さらにストレージノードを追加する必要はありません。
- このアーキテクチャでは、一時的な Amazon EMR クラスターを使用できます。データは Amazon S3 に保持されるため、データ処理ジョブの完了後、Amazon EMR クラスターをシャットダウンできます。
- HDFS ノードをオーバーロードすることなく、同じデータセットで複数の Amazon EMR クラスター内の複数のジョブを実行できます。

また、このアーキテクチャには、少なくとも 1 つの潜在的な欠点があります:

- まず HDFS にデータをコピーしなければならないため、データ処理ワークフローに遅延が生じます。しかし、反復ワークロードでは、データ処理速度が高速であるため、遅延を補うことができます。

パターン 3: HDFS とバックアップストレージとしての Amazon S3

このアーキテクチャパターンでは、HDFS に直接データを格納し、Amazon EMR ノードがローカルにデータを処理し、S3DistCp または DistCp を使用して、Amazon S3 に定期的にデータをコピーします。このパターンの主な利点は、まず Amazon EMR ノードにデータをコピーすることなく、データ処理ジョブを実行できることです。

このアーキテクチャパターンではデータ処理速度は向上しますが、データの耐久性が主な欠点の 1 つです。Amazon EMR はデータの格納にエフェメラルディスクを使用するので、Amazon EMR EC2 インスタンスが異常終了した場合、データが失われる恐れがあります。HDFS 上のデータは Amazon EMR クラスター内でレプリケーションされ、ノード障害が発生しても通常 HDFS は復旧できますが、レプリケーション係数よりも損失したノードの数が多い場合、データが損失する恐れがあります。データの損失を避けるために、DistCp または S3DistCp を使用して定期的なフェーズで HDFS データを Amazon S3 にバックアップすることをお勧めします。また、既存の Amazon EMR クラスターに過度の負荷がかからないようにするために、データセット全体ではなくデータパーティションのバックアップを作成しておくこともお勧めします。

パターン 4: 伸縮自在な Amazon EMR クラスター（手動）

手動パターンでは、Amazon EMR アーキテクチャは毎日のデータフローを処理できるだけのノードから始まります。時間の経過に伴いデータは増大するため、容量を手動でモニタリングし、毎日の処理ニーズに合わせて Amazon EMR クラスターのサイズを事前に設定する必要があります。

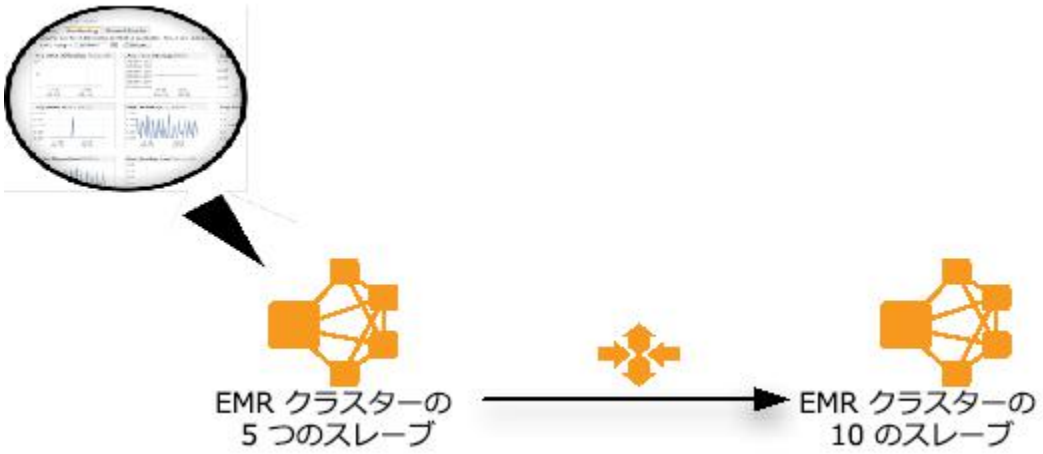

図 16: Amazon Cloudwatch を使用した Amazon EMR 容量のモニタリング

Amazon EMR はいくつかの Amazon CloudWatch メトリックスを使用します:

- 実行中のマッパーの数
- 未処理のマッパーの数
- 実行中のリデューサーの数
- 未処理のリデューサーの数

このアーキテクチャでは、Amazon EMR クラスターは最初毎日のデータ量を処理できるだけの x 個のノードから始まります。ただし、上記の Amazon CloudWatch メトリックスをモニタリングした結果、Amazon EMR クラスター内の未処理のマッパーまたはリデューサーの数が多くなり、Amazon EMR 処理時間が長くなっていることに気づきます。つまり、データセット全体を処理するだけの Amazon EMR 容量が足りません。

データの増大への対処は共通の問題です。この解決として、API、CLI を使用、または AWS を介して手動でノードを追加できます。次のセクションでは、動的で伸縮自在な Amazon EMR アーキテクチャを定義します。

パターン 5: 伸縮自在な Amazon EMR クラスター（動的）

Amazon EMR は 3 つのノードタイプから構成されます:

- マスターノード（JobTracker および NameNode を実行）
- コアノード（TaskTracker および DataNodes を実行）
- タスクノード（TaskTracker のみを実行）

コアノードは HDFS ストレージ（DataNode）を提供するのに対し、タスクノードは提供しません（ローカルな DataNode でありません）。タスクノードはデータ処理専用です（マッパーおよびリデューサーの実行）。HDFS がタスクノードにないことの利点の 1 つは、Amazon EMR クラスターがタスクノードを簡単に追加または削除できることです。これらのノードで HDFS が動作しないため、不要なノードを削除しながら、クラスターにタスクノードを追加して、処理能力を上げることができます。

このパターンでは、Amazon EMR クラスターは当初限られた数のコアノードから構成され、最低限の HDFS ストレージを提供します。Amazon EMR がデータ処理ジョブをクラスターに発行すると、このクラスターにタスクノードを追加し、コアノードが提供するもの以上に計算能力を高めることができます。データ処理ジョブが完了したら、Amazon EMR タスクノードの削除を開始できます。

Amazon EMR クラスターにタスクノードを追加するジョブは、Amazon EMR CloudWatch メトリックスを使用して自動化できます。Amazon EMR はいくつかの CloudWatch メトリックスを使用できます:

- 実行中または未処理のマッパーの数
- 実行中または未処理のリデューサーの数
- アイドル状態のクラスター
- ライブ状態のデータノードまたはタスクノード

Amazon CloudWatch と Amazon Simple Notification Service（Amazon SNS）の連携により、Amazon CloudWatch メトリックスにアラームをセットアップし、各アラームしきい値で通知を受信できます。Amazon EMR CloudWatch メトリックスを SNS 通知と組み合わせることで、Amazon EMR クラスターの容量がなくなった場合にさらにタスクノードを追加するなどのアクションを自動化できます。例えば、永続的な Amazon EMR クラスターを組織内の複数のグループで共有し、それぞれのグループが固有のデータ処理ジョブを Amazon EMR に送信することができます。CloudWatch メトリックスをモニタリングすると、業務時間中は多くのジョブが発行され、その多くがキューに登録するのに長い時間がかかるため、Amazon EMR クラスターの処理速度が低下することに気づきます。下記のステップに、クラスターに Amazon EMR ノードを動的に追加するために実装できる自動ワークフローの概要を示します:

1. Amazon EMR が CloudWatch メトリックスを公開します。しきい値に達したときに通知するように、Amazon EMR CloudWatch メトリックスで CloudWatch アラームをセットアップできます。
2. Amazon EMR は、EC2 インスタンスまたは AWS Elastic Beanstalk でホストされる HTTP エンドポイントに SNS 通知を送信します。HTTP エンドポイントは、AWS SNS から HTTP 通知を取り出し、適切なアクション（ステップ 3）をトリガーするシンプルなアプリケーション（Java servlet）です。（**注意:** HTTP エンドポイントは顧客が開発したアプリケーションです。）
3. HTTP エンドポイントアプリケーションは Amazon EMR API エンドポイントにリクエストを行い、現在実行中の Amazon EMR クラスターにさらにインスタンスを追加します。
4. Amazon EMR ワークフロー管理は、Amazon EMR クラスターにさらにノードを追加します。

注意: CloudWatch アラームが低利用率アラートをトリガーすると似たようなワークフローが適用され、使用していない Amazon EMR ノードの削除が開始されます。

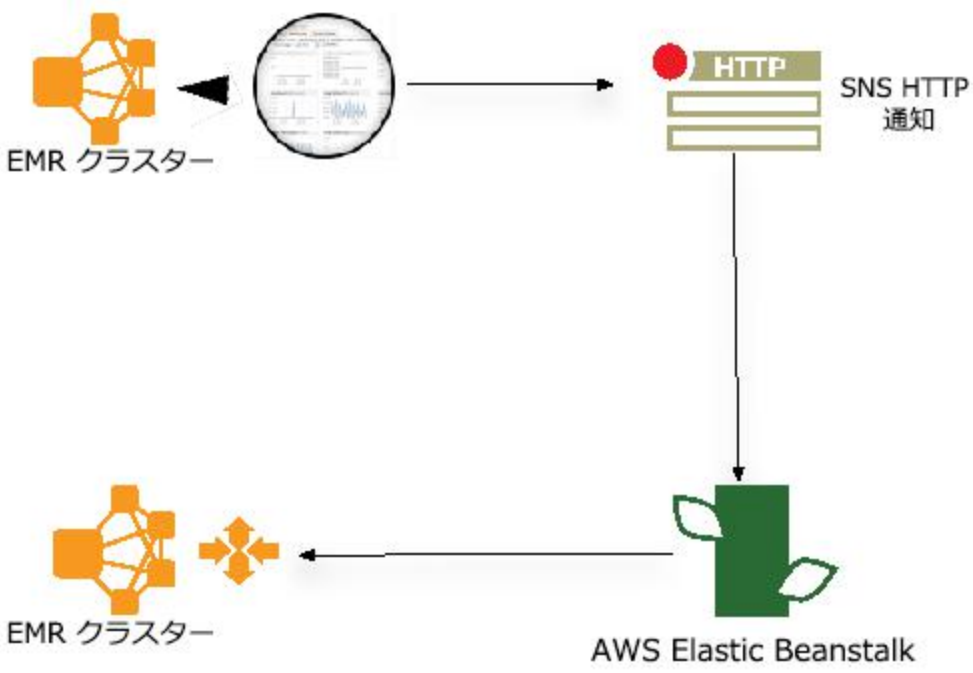

図 17: Amazon EMR 容量のプロビジョニングの自動化

このアーキテクチャパターンには、いくつかの重要な利点があります:

- 多くの計算能力が必要な場合にのみノードを追加し、使用していないときは削除することで、コストを抑えることができます。
- ほとんどの Hadoop アーキテクチャでは、Hadoop ノードが提供するリソースに限られます。Amazon EMR を使用することにより、必要に応じてさらに計算能力を追加することにより、処理時間を制御および短縮できます。

Amazon EMR および Amazon EC2 でのコストの最適化

AWS は、コンピューティングコストを最適化するためのさまざまな方法を提供します。通常、AWS は 3 つの価格モデルを提供します:

1. **オンデマンドインスタンス** – オンデマンドインスタンスを使えば、使用する処理能力に対して、時間単位で料金をお支払いいただきます。長期間の契約は不要です。
2. **リザーブドインスタンス (RI)** – リザーブドインスタンスとは、予約したい各インスタンスについて、事前に低額の予約金を支払うオプションを提供し、その見返りとして、そのインスタンスについて、時間あたりの使用料金に対して大幅な割引を受け取るというものです。
3. **スポットインスタンス** – スポットインスタンスを使えば、使用していない Amazon EC2 容量について入札を行い、EC2 の時間あたりの使用料金を割り引き価格で支払うことによりコンピューティングコストを低減できます。

Amazon EMR ワークロードに応じて、EC2 リザーブドインスタンスまたはスポットインスタンスを購入することで、Amazon EMR のコストを最適化できます。オンデマンドインスタンスは、一時的な Amazon EMR ジョブがある場合、または Amazon EMR の 1 時間あたりの使用量がその時間の 17% 未満である場合に適したオプションです。ただし、Amazon EMR の時間あたりの使用料金が 17% を超える場合は、リザーブドインスタンスのほうが経済的です。リザーブドインスタンスでは、予約するインスタンスごとに低額の一括払いオプションを提供し、その見返りとして、そのインスタンスについて、時間当たりの使用料金に対して大幅な割引を受け取ります。

AWS では、EC2 インスタンスの使用量に基づいて、3 つのタイプ（軽度使用、中度使用、重度使用）のリザーブドインスタンスがあります。Amazon EMR クラスターの使用率が判明している場合は、さらに経済的です。軽度使用モデルは、1 日に 2 時間だけまたは 1 週間に数日だけ定期的に実行する Amazon EMR ワークロードの場合に適したオプションです。中度使用リザーブドインスタンスは、この数年 Amazon EC2 が提供しているのと同じリザーブドインスタンスです。Amazon EMR ワークロードを常時実行するわけではなく、必要でない場合は Amazon EMR クラスターをシャットダウンするオプション（一時的な Amazon EMR ワークロード）が望ましい場合に適したオプションです。一貫して定常状態の Amazon EMR ワークロードを実行する場合は、重度使用モデルが適しています。

いくつかの例を見てみましょう。

例 1

1 時間ごとに処理ジョブを実行する安定的な日次 Amazon EMR ワークロードがあり、ジョブの完了時にクラスターをシャットダウンせずに実行状態にしておく（永続的）と有益であるとします。以下のグラフは、架空の日次 Amazon EMR ワークロードを表します。青いエリアは、ジョブを処理するために使用しているインスタンスの 1 時間あたりの数を表します。時間の 100% でクラスターが使用されていることが明白です。ワークロードが前述した特性と一致している場合、重度使用リザーブドインスタンスが経済的です。

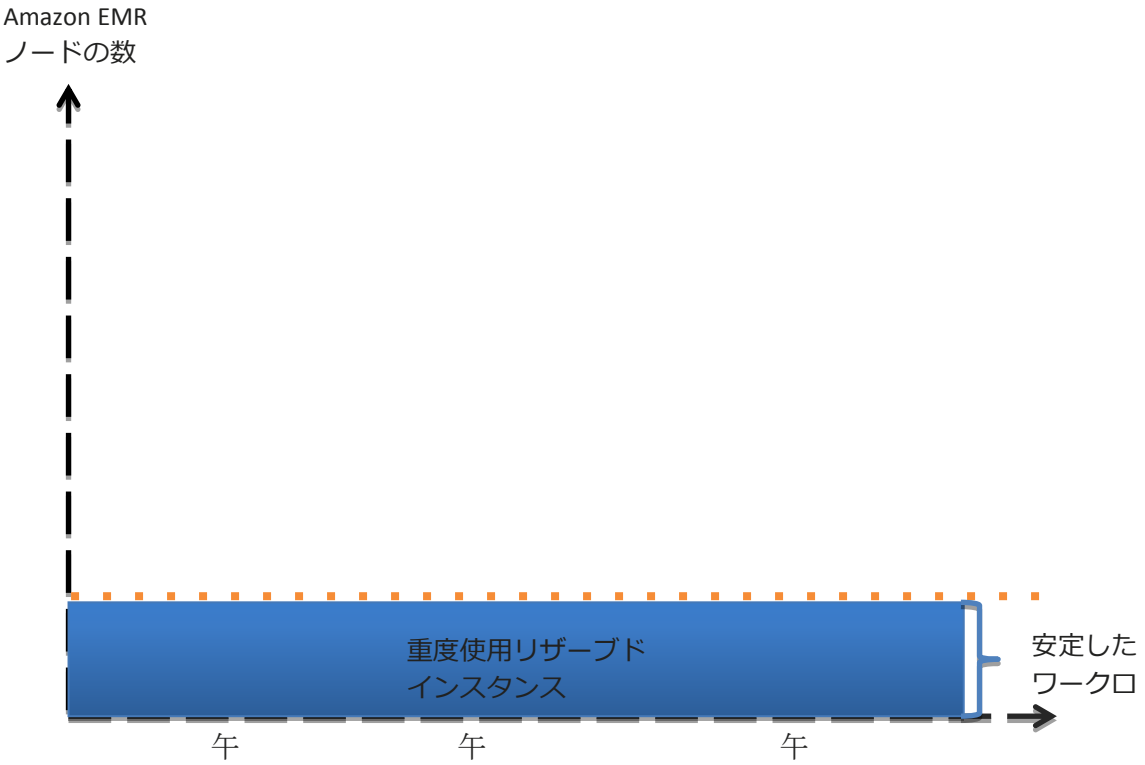

図 18: 重度使用リザーブドインスタンスを使用したコストの最適化

例 2

1 日に数度処理ジョブを実行する Amazon EMR ワークロードがあるとします。このシナリオでは、永続的なクラスターを必要とするだけの Amazon EMR クラスターを使用していないため、処理後 Amazon EMR クラスターをシャットダウンします。時間の 100% でクラスターを使用しているわけではないため、重度使用リザーブドインスタンスは最適な選択肢ではありません。代わりに、中度使用リザーブドインスタンスを使用したほうが経済的です。中度使用リザーブドインスタンスは重度使用リザーブドインスタンスほど経済的でないものの、処理ジョブが時間の 100% を消費していないワークロードにとっては最適なオプションです。さらに、オンデマンドインスタンスと比べ、より望ましい料金を実現します。

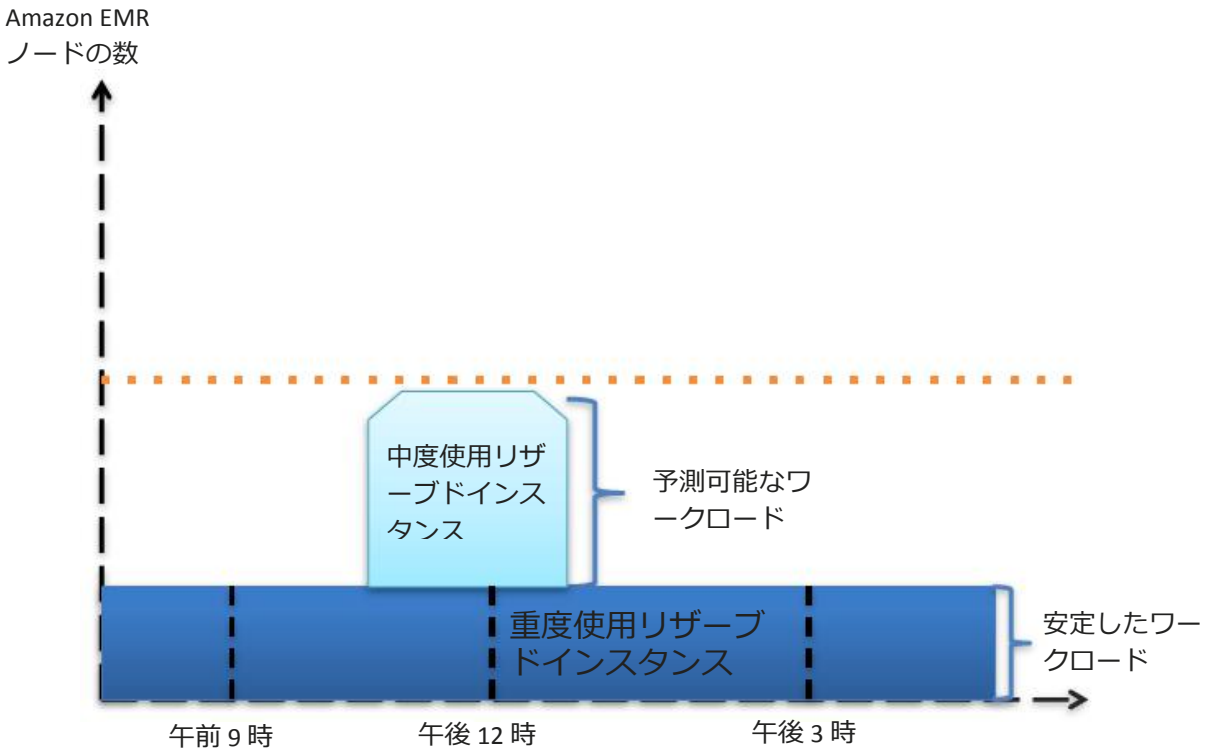

図 19: リザーブドインスタンスを使用したコストの最適化

例 3

上記の 2 つのモデルでは、ワークロードが安定状態（1 時間単位）であるか、または 1 日に限られた回数であることが前提でした。ただし、両方のワークロードの混在もよく見られます。このシナリオでは、Amazon EMR クラスターを時間の 100% で必要とするワークロードと、予測可能または予測不能いずれかの一時的なワークロードがあります。予測不能なワークロードで推奨される価格モデルはスポットまたはオンデマンドです。このシナリオをわかりやすくするために、以下の Amazon EMR の時間あたりの使用料金を見てください:

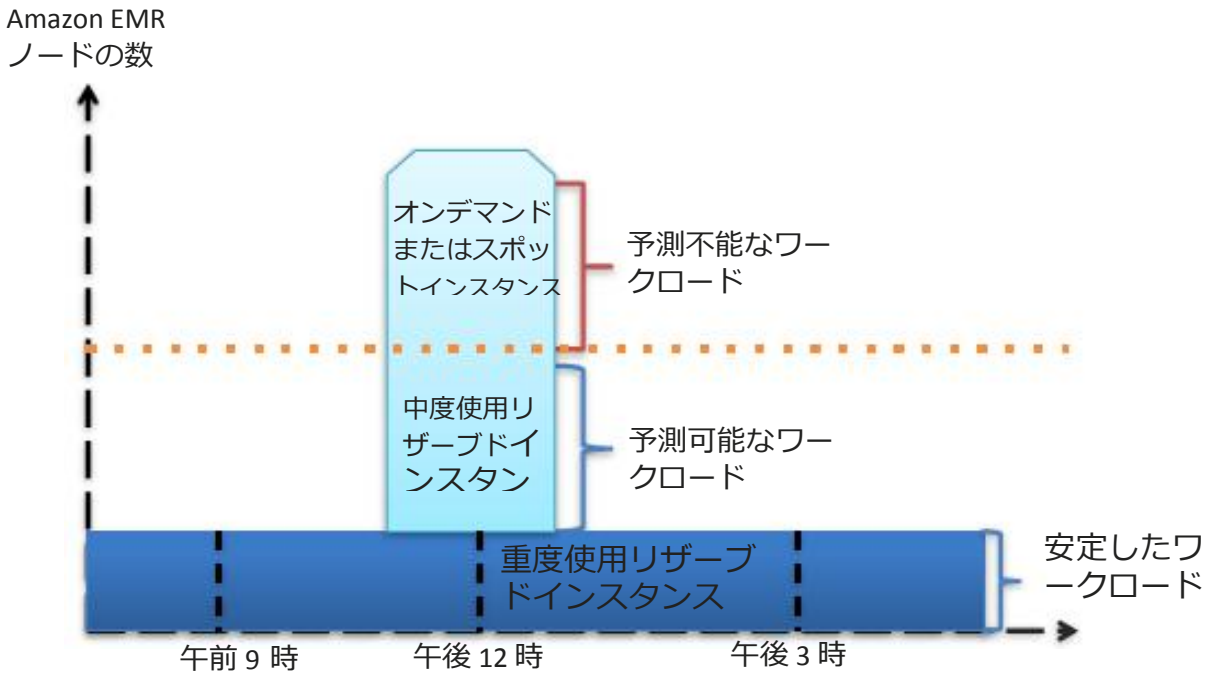

図 20: オンデマンドおよびスポット料金表を使用したコストの最適化

EC2 スポットインスタンスでのコストの最適化

スポットインスタンスとは、Amazon EC2 の処理能力の価格をお客様が指定できるシステムです。お客様は予備の Amazon EC2 インスタンスの価格を入札し、入札価格がその時点のスポット価格を上回っている場合に、インスタンスを実行できます。スポット価格は、需要と供給の関係に基づいてリアルタイムで変動します。スポットインスタンスの価格体系は、オンデマンドインスタンスとリザーブドインスタンスの価格体系を補完するものであり、アプリケーションによっては、計算処理能力を達成するうえで最もコスト効果の高い選択肢となる可能性があります。

スポットインスタンスは、時間の制約がなく、中断してもかまわないタスクの計算コストを大幅に削減できます。スポット価格は多くの場合、同じ EC2 インスタンスタイプのオンデマンド価格を大幅に下回ります（現在のスポット価格（http://aws.amazon.com/ec2/spot-instances/#7）を参照）。Amazon EMR で、Amazon EMR クラスターを EC2 スポットインスタンスで実行できます。EC2 スポットインスタンスを使用する場合にいくつかの検討事項とアークテクチャパターンを考慮する必要があります:

1. スポットインスタンスを使用するかどうかは、EC2 スポットインスタンスの入札価格によって決まります。入札金額が低すぎ、EC2 インスタンスの市場価格が入札価格を上回る場合、EC2 インスタンスは使用できなくなります。このため、長期間 EC2 インスタンスを保持できるような入札価格を必ず使用するようにしてください。EC2 スポット価格の履歴を調べ、スポットの価格パターンを検出することで、これを実現できます。スポットインスタンスに入札するためのもう 1 つの方法は、オンデマンド価格に近い値を付けることです。これにより、EC2 スポットを迅速に入札できるだけでなく、EC2 スポットインスタンスの存続期間を延ばすことにもなります。
2. Amazon EMR インスタンスは、マスターノード、コアノード、タスクノードの 3 つのカテゴリーに分類されます。Amazon EMR クラスター全体を EC2 スポットインスタンスで実行できます。ただし、マスターノードを失った場合（スポットの市場価格が入札価格を上回ったことにより）、クラスター全体を失うことになるため、スポットインスタンスでマスターノードを実行することはお勧めできません。

同じように、コアノードを EC2 スポットインスタンスで実行できます。ただし、コアノードは HDFS（データノード）を実行するため、コアノードを失った場合、Amazon EMR クラスターはその失ったデータを回復し、HDFS クラスターのバランスをとり直す必要があります（Amazon S3 の代わりに HDFS を使用している場合）。また、コアノードの大半を失い、他の使用可能なノードから HDFS データを回復できなくなった場合、クラスター全体を失うことになることを忘れないでください。

タスクノードは EC2 スポットインスタンスで実行するのに最も安全なノードです。タスクノードはデータ（非 HDFS）をホストしないため、市場価格の高騰によりタスクノードを失った場合、Hadoop TaskTracker ノードを失うことを意味します。TaskTracker ノードを失うと、Hadoop JobTracker はタスク処理障害（マッパーまたはリデューサーのタスク）を検出し、障害が発生したタスクを別の Amazon EMR ノードで再試行します。これによってデータ処理速度が低下する可能性がありますが、Amazon EMR クラスターの全体的な状態には影響ありません。

3. クラスター全体をスポットインスタンス（マスターノードおよびコアノード）で実行するリスクはあるものの、データが Amazon S3 にあれば、Amazon EMR クラスター全体をスポットインスタンスで実行できます。このため、マスターノードまたはコアノードを失い、最終的に Amazon EMR クラスター全体を失った場合でも、新しい Amazon EMR クラスターを起動することで、データ処理ジョブを再度実行できます。この結果ジョブに遅延が発生するものの、その遅延を許容できれば、非常に経済的です。
4. Amazon EMR データ処理ジョブでスポットインスタンスを使用することを決定した場合は、以下のアーキテクチャを検討してください。オンデマンドインスタンスまたはリザーブドインスタンスでマスターノードを実行します（永続的な Amazon EMR クラスターを実行している場合）。オンデマンドインスタンスまたはリザーブドインスタンスを使用して Amazon EMR クラスターの一部をコアノードで実行し、残りのクラスターをスポットインスタンスを使用してタスクノードで実行します。例えば、ジョブを処理するために 20 の Amazon EMR ノードが必要であることがわかっている場合は、オンデマンドインスタンスまたは RI インスタンスを使用してコアインスタンスで 10 のノードを実行し、残りの 10 のノードをスポットインスタンスを使用してタスクノードで実行します。

パフォーマンスの最適化（高度）

本書で説明する最適化に時間を費やす前に、各インスタンスで得られる Hadoop 構成について Amazon EMR チームがすでに厳密な最適化（そのインスタンスで使用可能な CPU およびメモリリソースに基づく）をすでに実施していることに留意してください。また、本書に示す最適化は上級の Hadoop ユーザーにのみ推奨します。Hadoop を使い始めたばかりの場合は、これはお勧めしません。

全体的にみると、パフォーマンスの向上によりコストが最適化される場合を除き、通常はパフォーマンスよりもコストの最適化をお勧めします。その理由は以下のとおりです:

- 最適なパフォーマンスの最適化は、適切なデータ構造（つまり、スマートなデータのパーティション化）です。データを効率的に構造化することにより、Hadoop が処理するデータの量を制限でき、これは本質的にパフォーマンスの向上につながります。Amazon EMR クラスターはインスタンスタイプに基づいてすでに最適化されているため、本書で後述する他の最適化ではごくわずかしかパフォーマンスを向上できません。ただし、3 桁のクラスターサイズの場合は例外であり、処理時間を数秒または数分短縮するだけで、処理時間およびコストをかなり低減できます。

- Hadoop は、通常の処理時間を時間単位または日数単位で測定するバッチ処理フレームワークです。処理時間に制約がある場合は、Hadoop は適切なフレームワークでない可能性があります。言い換えると、データ処理の SLA を満たすために数分だけ処理時間を短縮する必要がある場合、Twitter の Storm や Spark などの他のフレームワークのほうが要件を適切に満たすことができます。
- Amazon EMR は 1 時間単位で料金を請求します。つまり、クラスターを一度実行すると、その 1 時間分の料金を支払います。これは忘れないでください。Amazon EMR クラスターの 1 時間分の料金を支払う場合、データ処理時間を数分短縮しても、それに費やした時間や作業に見合わない可能性があります。
- パフォーマンスを向上させるためにより多くのノードを追加することは、クラスターの最適化に時間をかけるよりも安価で済むことを忘れないでください。

Amazon EMR クラスターの最適化を決定した場合、以下のステップをお勧めします:

1. 最適化の前後にベンチマークテストを実行してください。
    a. ベンチマークが毎日のワークロードを反映するようにしてください。
    b. できる限り多くの変数を除去してください。例えば、CPU およびメモリの使用率を最適化する場合、Amazon S3 をデータソースとして使用せず、ベンチマークを実行する前に、HDFS にデータを移動します。CPU およびメモリの使用率に満足したら、Amazon S3 にあるデータを使用して、ベンチマークを実行します。
    c. ベンチマークを複数回実行して、その平均処理時間をベースラインとして使用してください。
2. Ganglia を使用してベンチマークテストをモニタリングします。Ganglia はオープンソースのモニタリングツールで、ブートストラップアクションを使用して Amazon EMR にインストールできます。[6]
3. Ganglia のメトリックスをモニタリングすることで、制約を識別します。通常制約が検出されるのは以下のエリアです:
    - メモリ
    - CPU
    - ディスク I/O
    - ネットワーク I/O
4. 潜在的な制約を識別したら、最適化により制約を削除し、新しいベンチマークテストを開始します。

パフォーマンス向上のための提案

マッパーとリデューサーはどちらもパフォーマンスを最適化するためのエリアです。

マップタスクの向上

マッパータスクを最適化する上で役立つヒントを下記に示します。

- **マップタスクの存続期間** – Hadoop マッパータスクをモニタリングして、平均実行期間を監視します。寿命が短い場合（例えば、分単位ではなく秒単位など）、マッパーの数を減らすことができます。マッパーの数は入力サイズ（分割）の関数であるため、マッパーの数を減らすことは、通常、大きなファイルサイズを使用することを意味します。 言い換えると、マップタスクの寿命が短い

[6] http://ganglia.sourceforge.net/

場合、小さいファイルを処理している可能性があり、これを大きなファイルに集約することにより利益を得ることができます。2 つの手法があります:

i. ファイルサイズ別または時間別にファイルを集約します。

ii. 小さなファイルを大きな Hadoop アーカイブファイル（HAR）に集約します。

- **マッパー出力の圧縮** – 圧縮は、ディスクに書き込まれるデータ量が少なくなり、ディスク I/O が向上することを意味します。`FILE_BYTES_WRITTEN` Hadoop メトリックスを調べることで、ディスクに書き込まれるデータの量をモニタリングできます。また、圧縮は、リデューサーがデータを取得するシャッフルフェーズで役立ちます。クラスターの HDFS データレプリケーションにも役立ちます。

  `mapred.compress.map.output` を `true` に設定することで、圧縮を有効にします。圧縮を有効にすると、圧縮アルゴリズムも選択できます。LZO はパフォーマンスが高く、圧縮および解凍が高速です。

- **マップタスクのディスク書き出しの回避** – マッパーがバッファ内にデータを保持するのに十分なメモリを提供することにより、マップタスクがディスクに書き出し、または書き込まれないようにします。`io.sort.*` パラメーターを増やすことで、これを行うことができます。Amazon EMR チームは、使用しているインスタンスタイプに基づいてこの値を設定しています。ほとんどの場合、マッパーのタスクバッファのサイズを決定する `io.sort.mb` パラメータを調整します。`SPILLED_RECORDS` という Hadoop メトリックスをモニタリングし、`io.sort.mb` 値を大きくしてみます。ただし、この数値をあまり高く設定すると、マッパー処理のメモリが制限され、Java メモリ不足エラーが発生する可能性があるため、注意してください。通常、`io.sort.mb` を増やす前に、マッパーヒープサイズを増やします。

リデュースタスクの向上

リデューサータスクを最適化するには、以下のヒントが役立ちます。

- ジョブでは、クラスターのリデューサーの合計容量よりも少ないリデューサーを使用してください。リデューサーは同時に完了するようにして、リデューサーに実行を待たせないようにしてください。クラスターの容量よりも多くのリデューサーを使用するジョブの場合、リデューサーが実行を待機することになります。CloudWatch は **Average Reduce Task Remaining（残りのリデュースタスクの平均）** や **Average Reduce Tasks Running（実行中のリデュースタスクの平均）** などの Hadoop メトリックスを提供します。最高のパフォーフォマンスを達成するには、クラスターの大きさが、すべてのリデュースタスクが「実行」状態で、「残り」状態がゼロになるようにしてください。
- マッパーの構成と同じように、できる限り多くのデータをメモリ内に保持します。リデューサージョブで必要とされるメモリフットプリントが小さい場合は、以下のパラメータを調整することで、リデューサーメモリを増やします:
  - `mapred.inmem.merge.threshold`（`0` に設定）
  - `mapred.job.reduce.input.buffer.percent`（`1.0` に設定）

Ganglia を使用したパフォーマンスの最適化

Ganglia を使用している場合は、データ処理のパフォーマンスを向上させるために以下のヒントが役立つかもしれません。

- **CPU** – ジョブを実行し、クラスターの CPU 使用率を確認します。CPU をすべて使用しているわけではない場合、ノードあたりのタスク容量を増やし（マッパーまたはリデューサーの容量を増やし）、Amazon EMR クラスター内のノードの数を減らすことができます。これが有効であるのは、CPU をすべて使用していない場合だけであることを忘れないでください。CPU が最大限使用されている（ジョブが CPU の制約を受けている）場合、各ノードにさらにタスク容量を追加すると、CPU のコンテキスト切り替えが増え、パフォーマンスが低下する恐れがあります。
- **メモリ** – Ganglia を使用してメモリの使用率を確認します。ジョブの実行後、メモリがすべて使用されていないことに気づいた場合、タスクジョブ（マッパーまたはリデューサー）に使用可能なメモリ量を増やすことで利益が得られることがあります。例えば、以下の例では、CC2 インスタンスあたりのマッパーの量は減りますが、マッパー/リデューサーのメモリは増えます:

```
elastic-mapreduce --create --alive --instance-group master --
instance-type c1.xlarge --instance-count 1 --instance-group core --
instance-count 25 --instance-type cc2.8xlarge --bootstrap-action
s3://elasticmapreduce/bootstrap-actions/configure-hadoop --args "-
m,mapred.tasktracker.map.tasks.maximum=14,-m,mapred.child.java.opts=-
Xmx2300m"
```

- **ネットワーク I/O** – Amazon S3 と Amazon EMR を使用している場合、Ganglia をモニタリングして、ネットワークのスループットを監視します。できるだけ多くの入力ファイルを並列でダウンロードするのに十分なマッパーを各ノードで確保することにより、NIC スループットを最大限高めることが目標です。例えば、処理しなければならないファイルが Amazon S3 に 100 ファイルある場合、すべての入力ファイルを並列処理するには Amazon EMR クラスターに合計 100 マッパーの容量が必要です。クラスターに 100 マッパーの容量がない場合、以下の 2 つのオプションが役立ちます:
    i. 最も簡単なオプションは、Amazon EMR クラスター内のノードの数を増やすことです。
    ii. もう 1 つのオプションはノードあたりのマッパー容量を追加することです。例えば、ノードあたりのデフォルトのマッパー容量が 2 の場合、もっと大きな数値に増やすことができます。さらにマッパーを追加する前に、Ganglia を監視して、ジョブが CPU 制約を受けていないことを確認することが重要です。ジョブですでに CPU の 100% を使用している場合は、マッパーを追加しても効果ありません。ただし、CPU の制約を受けず、ノードあたりさらにマッパーを追加できる場合、マッパータスクが増えたことで、Amazon S3 への並列ダウンロードスレッドが追加され、これによってマッパーの処理時間が短縮されます（各マッパーは、データをダウンロードするための Amazon S3 への 1 つのスレッドに対応します）。CPU または NIC スループットをすべて使用するまで、このプロセスを繰り返すことができます。もう 1 つモニタリングすべき項目はメモリです。ノードあたりのマッパー容量を増やす場合は、新たに追加するマッパーをサポートするのに十分な空きメモリがあることを確認してください。

クラスター内に十分なマッパー容量があるにもかかわず、マッパー容量よりも少ないマッパーしかジョブで実行されていないという場合があります。顕著なケースは、データが必要とするよりも多くのノードが Amazon EMR クラスターにある場合です。もう 1 つ潜在的なケースは、分割できない大きなファイル（GZ で圧縮されたファイルなど）を使用している場合です。この場合の潜在的な最適化は、圧縮ファイルを小さいファイルに分割してデータファイルのサイズを小さくするか、または分割をサポートする圧縮を使用することです。ファイルをどの程度小さくまたは大きくするかバランスを取る必要があります。クラスターのマッパー容量をすべて使用するまでファイルを分割するのが最適です。

- **JVM のモニタリング** – Ganglia JVM メトリックスをモニタリングし、GC（ガベージコレクター）の一時停止がないか監視します。GC の長い一時停止がある場合は、マッパー/リデューサータスクに十分なメモリが提供されていないことを意味します。できる限り、JVM メモリを増やします。増やすことができない場合は、インスタンスを追加して、負荷を解消します。
- **ディスク I/O** – 過剰なディスク I/O に注意してください。ディスク I/O は大きなボトルネックになる可能性があるため、できる限りディスクへ書き込まないようにしてください。ディスク I/O を最適化する上で、マッパーの書き出しとリデューサーの書き出し（`SPILLED_RECORDS`）の 2 つの設定が役立ちます。Hadoop メトリックスでこの両方をモニタリングできます（以下の「Hadoop メトリックスの検出」セクションを参照）。ディスクの書き出しを減らすには、以下を試してください:
    i. マッパー出力に圧縮を使用します: `mapred.compress.map.output` を `true` に設定します。
    ii. Ganglia でディスク I/O メトリックスをモニタリングし、必要に応じてタスクメモリを増やします。上記の圧縮に関する推奨事項に従い、ディスクアクティビティと Hadoop タスクメモリが大幅に改善された場合、Hadoop のバッファサイズを増やすことで、ディスク I/O のパフォーマンスを向上できます。デフォルトでは、Hadoop バッファは小さい値（4 KB）に設定されています。`io.file.buffer.size` パラメータを変更することで、この数値を増やすことができます。

Hadoop メトリックスの検出

Hadoop JobTracker Web UI を使用すると、Hadoop の内部メトリックスに簡単にアクセスできます。各ジョブの実行後、Hadoop はジョブの特徴を理解する上で非常に有効ないくつかのメトリックスを提供します。こうしたメトリックスを検出するには、以下のステップに従います:

1. JobTracker UI インターフェース（masternode:9100）に移動します。
2. 完了済みジョブまたは実行中のジョブをクリックします。
3. 全体的な集約メトリックスを検討します。
4. マッパー/リデューサーをクリックして、そのマップリデュースジョブのメトリックスを取得します。

まとめ

Amazon EMR を使用することで、企業は短期間で Hadoop クラスターを簡単にデプロイおよび運用できます。本書では、アマゾン ウェブ サービス（AWS）にデータを移動し Amazon Elastic MapReduce（EMR）で処理するためのベストプラクティスとアーキテクチャパターンをいくつか紹介し、データの集約および圧縮のベストプラクティスを確認するとともに、Amazon EMR クラスターを最適化するためのヒントを提供しました。

詳細資料と次のステップ

1. Getting Started With Amazon Elastic MapReduce – Video Series（http://aws.amazon.com/elasticmapreduce/training/）。
2. Amazon Elastic MapReduce Developer Guide（http://aws.amazon.com/documentation/elasticmapreduce/）。

Appendix A: Benefits of Amazon S3 compared to HDFS

- Amazon EMR と Amazon S3 の組み合わせの最大の利点の 1 つは、データの処理が完了すると、Amazon EMR クラスターをシャットダウンできることです。多くのワークロードがこの利点を利用できます。
- Amazon S3 上にデータを置くことで、Amazon EMR ワークロードをスポットインスタンスで実行でき、スポットインスタンスが失われることを心配しないで済みます。
- データは HDFS ノード障害から安全に守られます。Hadoop の高い係数は効果的ですが、HDFS ノードを失い、データの損失を招く場合もあります。Amazon S3 にデータを格納すれば、HDFS ノード障害から安全に守ることができます。
- Amazon S3 は、ライフサイクル、Secure Side Encryption、S3 Glacier など、セキュリティの向上やコストの削減のために利用可能な機能を備えています。
- Flume、Fluentd、Kafka など、一般的なサードパーティ製の分散ログコレクターを使用している場合、高スループットデータストリームを Amazon S3 に取り込むほうが HDFS に取り込むよりもはるかに簡単です。言い換えると、Amazon S3 をデータソースとすると、Amazon EMR は大量のデータを取り込むことができます。