IBM Cloud でより迅速なビッグデータ分析とより優れた応答性を

IBM Cloud 構成で、AWS のソリューションより短時間、高スループットでビッグデータ分析のワークロードを完了

クラウドが、ビッグデータ・イニシアチブの幅広い要求を満たすための拡張性と柔軟性をもたらします。しかし Infrastructure as a Service (IaaS) のプロバイダーがすべて同じというわけではありません。同時にビッグデータ・イニシアチブを成功に導くために、企業はパフォーマンスやスピードにおける優位性を求めています。

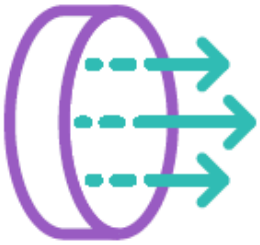
Principled Technologies は、2 つの IaaS プロバイダー (IBM® Cloud と Amazon® Web Services (AWS®) Elastic Compute Cloud® (EC2®)) で、Hadoop ベースのビッグデータ・ワークロードと一連のネットワーキングのテストを実施しました。データ・ポイントを仮想マシン(VM) 上にクラスタリングするスピード、およびビッグデータ処理中のスループットについて、IBM Cloud ソリューションが AWS ソリューションを上回りました。ビッグデータ分析は次のマーケティング・キャンペーン、アプリ、運用上の改善を推進します。IBM Cloud ソリューションは、この分析を AWS よりも短い待機時間で提供します。

さらに IBM Cloud ソリューションはデータセンター間のデータ転送や ping 送信でも AWS より短い時間で実行できました。米国、英国、日本にある各データセンター間のデータ送信が速いということは、世界各地にまたがる従業員によるコミュニケーション、計画、協業をより迅速に行えるということです。

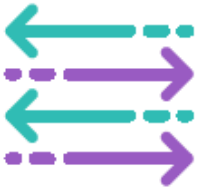
企業のビッグデータ・イニシアチブや世界規模のコミュニケーションは、その企業が利用している IaaS プロバイダーの影響を受けます。ビッグデータのパフォーマンスやスピードに関して言えば、IBM Cloud ソリューションは AWS ソリューションよりも、多くの利益を企業にもたらすことができているという調査結果が示されています。

ビッグデータからより迅速に価値ある洞察を引き出すのに IBM Cloud ソリューションが役立ちます

データのクラスタリング時間が 28% 短縮



1 秒当たりのクラスタリング量が 39% 増加



データの転送時間が最短 3 分の 1 に短縮

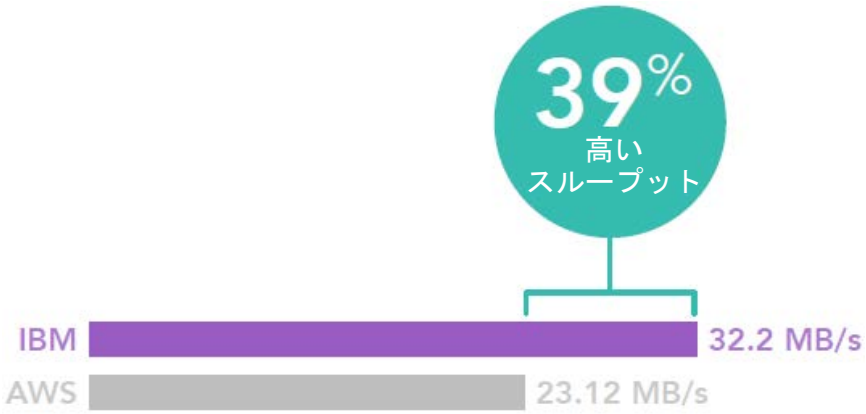
ビッグデータ・イニシアチブにクラウドを選択することで柔軟性と拡張容易性を実現

ビッグデータ分析は意思決定を推進したり、カスタマー・サービスや製品開発などの分野を改善したりする際に役立ちます。多数のデバイスやユーザーのホストから、絶えず増加し続けるデータ・セットを収集、ソート、分類、そして分割することでこれを実現します。これらのイニシアチブには、データの増加に対応するために、データのアナリスト、開発者やその他の IT スタッフがシームレスに作業内容を拡大できるインフラストラクチャーが必要です。クラウドを利用することで、このような拡張容易性や柔軟性の他に、以下のようなメリットも得ることができます。

- 注文処理の際に構成を選択する他は、設定する必要がほとんどありません。
- IT 管理者が追加で保守するオンサイトのハードウェアはありません。
- オンプレミスの計算リソースを追加できない、または追加したくない場合は、ビッグデータ分析を行うためのハードウェアの設備投資 (CapEx) や運用経費 (OpEx)(電力、冷却や管理) を負担する必要はありません。
- 短期間、または期間限定でビッグデータ分析を行いたい場合、通常はクラウドを使用する期間だけに料金を支払います。

より多くのデータで迅速な作業を

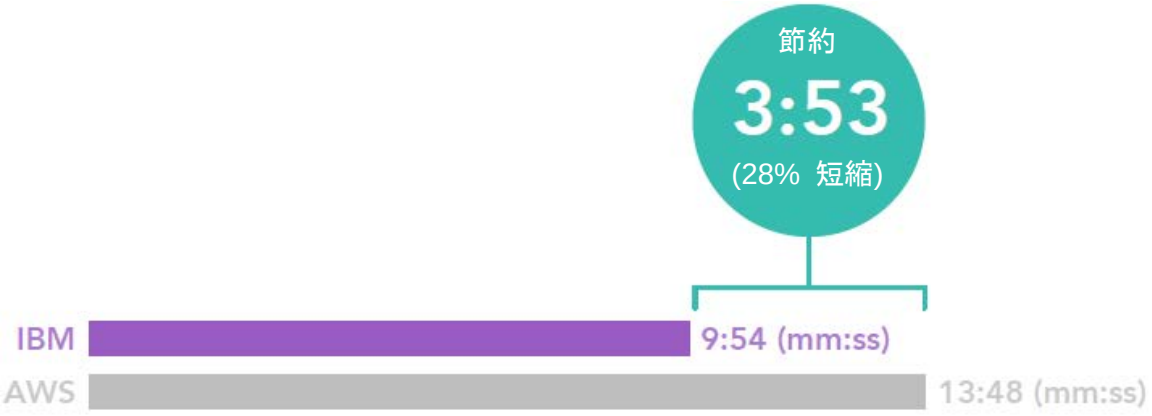
Principled Technologies では、ビッグデータ用ベンチマークである HiBench スイートの Kmeans ワークロードを実行しました。Kmeans とは、データ・ポイントをクラスタリングしてワークロードのスループットと所要時間をレポートする、機械学習のベンチマーク・ツールです。テスト結果を確認すると、スループットは IBM Cloud ソリューションが AWS 構成よりも 39% 優れていました (テスト方法について詳細は付録 B を参照してください)。使用している仮想化ソリューションが大量のスループットを実現すると、ビッグデータ・ワークロードを処理するために必要な VM は少ない数で済みます。このため大容量のデータ・セットを迅速に分析して処理するための運用経費 (OpEx) を削減できる可能性もあります。



1 秒当たりのデータ分析量が向上すると、アナリストはワークロードの所要時間をほとんど増やさず、より大きなデータ・セットに対応できるようになります。分析しなければならないデータ・セットが増加し続けても、IBM Cloud ソリューションを拡張して、より大きなビッグデータのワークロードに対応することができます。

より迅速にデータをクラスタリングして分析

効果的なマーケティング・キャンペーンを創出することに焦点を当てている場合でも、倉庫の運用効率性の改善を試みている場合でも、ソリューションでデータ・ポイントをクラスターにパーティション化するのが早ければ早いほど、データ・アナリストはより早く価値ある洞察やパターンを見極めることができるようになります。Kmeans テストで 18.7 GB のデータをクラスタリングする際、IBM Cloud ソリューションは AWS ソリューションよりも 3 分 53 秒 (28 %) 短い時間でこれを実行しました。テスト方法についての詳細は付録 B を参照してください。



洞察やパターンを即時に把握できるかどうかということが大切で、たとえそれが小さなデータの集合でも、お客様を一回限りでなくリピーターに変えるのか、収益を生み出す購買層を逃すかの分かれ道になります。ビッグデータ分析は、小売業だけでなく多くの業種で有用です。医療の分野でさえも、時間の節約が、適切な患者の診断や、治療法を繰り返し探して試すことにつながることがあります。運用上の観点から見ると、クラウドのスピードアップにより、企業はただワークロードを完了するためだけに VM をいくつも用意する必要がなくなります。

ワークロードによって選択できる IBM Cloud サービス

IaaS とクラウド・サービスのプロバイダーの中には、開発者やその他の IT スタッフが対応しなければならないような、限定的なサービスを提供しているものもあります。テストを行った IBM ソリューションは、ベアメタル、パブリック・クラウドやコンテナー・サービスを含む IBM Cloud の 170 を超えるサービスの一部です。コストとワークロード使用量のバランスを取るために、IT 部門と財務部門は価格オプションを 1 カ月単位や 1 時間単位で選択することができます。さらに、IBM Cloud は世界 19 カ国で所有・運営する 60 のデータセンターを持っているため、管理者はセキュリティーとコンプライアンスが保証される固有の場所でアプリケーションとワークロードの実行を選択することができます。[1] 加えて IT 管理者は、特定のワークロードに最適な計算リソースを幅広い Intel® Xeon® プロセッサーの中から選択できます。詳しくは、https://www.ibm.com/cloud-computing/jp/ja/ を参照してください。

地理的に離れたデータセンター間でも迅速にデータを転送

世界規模で展開されるビジネスでは、データのスピーディーな転送により、管理職やエグゼクティブは十分な情報に裏付けされた決断をタイムリーに下せます。米国、英国、そして日本の各データセンター間でデータを転送しましたが、IBM Cloud ソリューションは AWS よりもデータ転送が短時間で実行できました。IBM Cloud ソリューションでは、500MB のファイル転送時に最大の時間の節約が見られました。英国から米国へのデータセンター間のデータ転送では、IBM Cloud ソリューションは AWS よりも 2 分近く短時間でこれを行いました。英国から日本への IBM Cloud データセンター間の転送では、500MB のファイルで 3 分 30 秒以上の時間短縮としました。米国から日本へのファイル転送では、IBM Cloud ソリューションを使用した場合、AWS を使用した場合よりも 2 分 30 秒以上短縮されました。以下の図は、2 つのソリューションの所要時間を比較したものです。テスト方法の詳細については付録 B を参照してください。

ファイルとディレクトリーの転送 (秒)

25MB ファイル

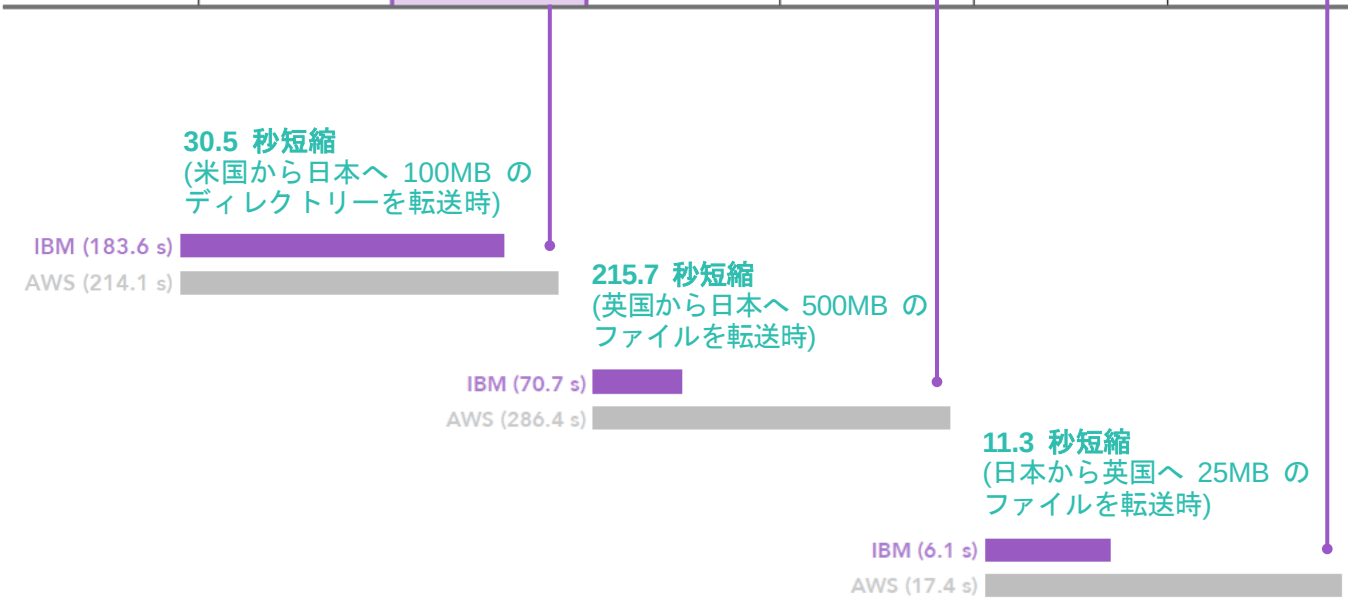
	米国→英国	米国→日本	英国→米国	英国→日本	日本→米国	日本→英国
IBM Cloud	2.5	4.1	2.5	6.2	4.1	6.1
AWS	5.8	11.4	6.1	16.7	11.3	17.4

500MB ファイル

	米国→英国	米国→日本	英国→米国	英国→日本	日本→米国	日本→英国
IBM Cloud	24.7	46.2	24.5	70.7	46.2	70.9
AWS	104.0	208.6	143.3	286.4	199.5	281.4

100MB ディレクトリー

	米国→英国	米国→日本	英国→米国	英国→日本	日本→米国	日本→英国
IBM Cloud	99.5	183.6	98.9	280.1	183.2	281.7
AWS	106.2	214.1	112.1	292.8	188.6	293



高速のデータ転送は、グローバル企業が運用上のボトルネックや生産性低下の可能性を減らすのに役立ちます。たとえば、英国、米国や日本にオフィスを構える金融機関では、市場や顧客に 24 時間 365 日体制での対応を求められる傾向にあります。

ファイル転送の他に、データセンター間での ping も行いました。行った ping テスト 6 回中のうち 5 回で、IBM Cloud ソリューションの方が優れた結果となりました。2 つのソリューションの最も大きな差は、米国と日本のデータセンター間で ping を送信した際に現れました。IBM Cloud ソリューションの方が約 18 ミリ秒速かったのです。ping の応答が速いということは、データセンター間の応答が速いということを意味します。以下の図表は、両ソリューションのテスト対象データセンター間で ping を通した際の違いを比較したものです。

ping テスト (ミリ秒)

	米国→英国	米国→日本	英国→米国	英国→日本	日本→米国	日本→英国
IBM Cloud	73.9	136.7	73.9	210.9	136.7	210.8
AWS	77.8	154.7	71.9	214.7	147.5	214.3

まとめ

ビッグデータ分析で大幅な組織改善が可能です。使用できるデータの迅速なクラスタリング、パーティション化が可能であれば、アナリストや意思決定を行う担当者は、ユーザーのニーズ、部門が効率的に運営されているか、また E メールを一斉配信するベスト・タイミングなどをほぼリアルタイムで理解することができるようになります。2 つのソリューションに Hadoop ベースの機械学習ワークロードやネットワーキングのテストを行ってきましたが、データのクラスタリング、スループット、そしてデータ転送の性能において IBM Cloud ソリューションが AWS ソリューションを上回っていることが分かりました。ビッグデータ分析の恩恵を受け、米国、英国と日本にあるオフィス間でデータを迅速に転送するために組織が必要とする IaaS プラットフォームは、IBM Cloud ソリューションであると言えるでしょう。

1 「Bluemix は IBM Cloud として生まれ変わりました。」2017 年 11 月 2 日、
https://www.ibm.com/blogs/solutions/jp-ja/bluemix-is-now-ibm-cloud/

2017 年 8 月 4 日、テスト対象のハードウェアおよびソフトウェアの構成を確定しました。ハードウェアとソフトウェアで現在リリースされているもの、および最近リリースされたものの更新は頻繁に行われるため、本レポートの公開時期に入手可能な最新バージョンと以下の構成は、同じでない場合があります。以前のシステムについては、標準的な内容で購入した場合の構成としてあります。実地テストは 2017 年 8 月 13 日に終了しました。

付録 A: システム構成情報

サーバー構成情報	AWS サーバー - Xen HVM domU
BIOS 名およびバージョン	Xen 4.2
オペレーティング・システム名およびバージョン/ビルド番号	Ubuntu 16.04.3 LTS
OS 更新/パッチの最終適用日	2017 年 7 月 20 日
プロセッサー	
プロセッサー数	1
ベンダーおよびモデル	Intel Xeon プロセッサー E5-2676 v3
コア数 (プロセッサー当たり)	40
コア周波数 (GHz)	2.40
ステップ実行	2
メモリー・モジュール	
システム内のメモリー合計 (GB)	160
メモリー・モジュール数	10
サイズ (GB)	16
ストレージ・コントローラー	
ベンダーおよびモデル	Intel 82371SB PIIX3 IDE
ローカル・ストレージ	
ドライブ #1 サイズ (GB)	100
ドライブ #2 サイズ (GB)	500
ネットワーク・アダプター	
ベンダーおよびモデル	Intel 82599 イーサネット・コントローラー

サーバー構成情報	IBM Cloud サーバー
オペレーティング・システム名およびバージョン/ビルド番号	Ubuntu 16.04.3 LTS
OS 更新/パッチの最終適用日	2017 年 7 月 20 日
プロセッサー	
プロセッサー数	1
ベンダーおよびモデル	Intel Xeon プロセッサー E5-2683 v4
コア数 (プロセッサー当たり)	32
コア周波数 (GHz)	2.10
ステップ実行	1
メモリー・モジュール	
システム内のメモリー合計 (GB)	128
ストレージ・コントローラー	
ベンダーおよびモデル	Intel 82371SB PIIX3 IDE
ローカル・ストレージ	
ドライブ #1 サイズ (GB)	350
ドライブ #2 サイズ (GB)	500

付録 B: テスト方法

比較シナリオを設計する

この分析のうち比較研究の部分は、私たちは IBM Cloud 構成と AWS 構成を対抗させ、直接、現実の環境で CPU 集約型のワークロード・パフォーマンスを測定しました。Intel HiBench Hadoop ベンチマーク・スイートの Kmeans Clustering ツールを利用しました。範囲の狭いタスクのみの CPU パフォーマンスを測定する総合的な (つまり客観的にあまり有用ではない) ベンチマーク・ツールではなく、平均的なコンピューターのスループットの差に焦点を当てるツールを選択しました。

コンピューターの各スループットを公平に評価するため、各クラウドのパフォーマンスが類似のものになるよう構成しました。ベンダーが提供する構成に制限があるため、完全に一致する構成にはできませんでした。以下のセクションに、主要な共通点と相違点をまとめます。

共通構成

各クラウドで、ノードは次の 4 つのうちいずれかの役割を果たします: クライアント (ベンチマーク・マシン)、構成 (構成管理ノード)、インフラストラクチャー (Hadoop ネーム・ノード。マスター・ノードとも呼ばれます)、またはスレーブ (Hadoop データ・ノード、すなわちテスト対象のシステムです)。クライアント、構成およびインフラストラクチャーの役割を果たす、4 コア、メモリー 16 GiB の VM を作成しました。各 VM には、ベンチマークと Hadoop のインフラストラクチャーをサポートするのに十分なストレージを割り当てました。

ESXi で最初の VM を作成

すべてのスレーブ・ノードは Intel Xeon プロセッサー E5-2600 v3 (仮想化済み) を使用しました。

IBM Cloud

- 各スレーブ・ノードには、32 スレッド (1 CPU x 32 コア/CPU x 1 スレッド/コア) および 128 GiB の RAM を用意しました。
- クラウド・ソリューションには 4 つのノード全体で 128 スレッド、合計 512 GiB の RAM を用意しました。

AWS

- 各スレーブ・ノードには、40 スレッド (1 CPU x 40 コア/CPU x 1 スレッド/コア) および 160 GiB の RAM を用意しました。
- クラウド・ソリューションには 3 つのノード全体で 120 スレッド、合計 480 GiB の RAM を用意しました。

ネットワーキング

すべてのノードは、ベンダーのプロビジョニング・オプションで許可され得る限り近接して同一の場所に置き、可能な限り高速のネットワーキングでプロビジョニングしました。IBM Cloud ソリューションについては、すべてのノードを 1Gbps ネットワークの VM で Dallas 06 に配置しました。AWS ソリューションについては、共通する配置グループを指定してノードを同一の場所に置き、それぞれ 10 Gbps ネットワークを割り当てました。

ストレージ

クライアント、構成およびインフラストラクチャーの役割については、汎用ストレージの 25GB のメイン・ドライブでノードをプロビジョニングしました。IBM Cloud ソリューションでは、ストレージ・エリア・ネットワーク (SAN) をベースにしたドライブを使用しました。AWS ソリューションは EBS (部分的に汎用 SSD (GP2)) で構成しました。

スレーブ・ノードについては、メインの OS ドライブにより大きなドライブと、Hadoop HDFS のサポートに十分な大きさのドライブをそれぞれ割り当てました。仮想 IBM Cloud ノードのドライブは、SAN ドライブ (OS に 100 GB、HDFS に 500 GB) です。AWS ソリューションのスレーブ・ノードは、3,000 IOPS IO1 ストレージ上で OS に 100 GB、HDFS に 500 GB を構成しました。

方法

以下の共通するコースに従い、両ソリューションのテストを行いました。

1. Linux Ubuntu 16.04 OS で VM をプロビジョニングする。
2. インフラストラクチャー・ノードとスレーブ・ノードに、Ambari を使用して Hadoop をインストールし構成する。
3. クライアント・ノードに、Intel HiBench をインストール・構成する。
4. クライアント・ノードに、IBM LPCPU 監視ユーティリティーをインストール・構成する。
5. HiBench ワークロードを準備する。
6. テスト用にスレーブ・ノードを調整する。
7. スレーブ・ノードで、 LPCPU モニタリングを開始する。
8. HiBench ベンチマークを実行し、結果を確認する。
9. スレーブ・ノードで、LPCPU モニタリングを停止し、結果を確認する。
10. 各レプリケーションで手順 5 から 9 を繰り返す。

Linux Ubuntu 16.04 での VM のプロビジョニング

各ベンダーのプロビジョニングの手段には若干の違いがありますが、共通するパターンに従っています:

1. Web ポータルにログインします。
2. デバイスまたはインスタンスをクリックします。
3. 以下の内容について、適切な値を選択します:
   - 数量
   - リージョンまたはプレースメント・グループ
   - オペレーティング・システム
   - CPU
   - メモリー
   - ネットワーキング
   - ディスク
4. リザベーション内容を確認し、VM の起動を待ちます。
5. SSH を使って VM にログインし、Hadoop、Ambari、HiBench、LPCPU、または使用する構成管理ソフトウェアをサポートするためのソフトウェア前提条件をすべて構成します。
6. クラスター内の各ホストについて、以下の手順を行います:
   a. ホスト <XXX> に接続します: > ssh root@XXX
   b. root ユーザー用に RSA 鍵ペアを作成します: > ssh-keygen -t rsa -b 8192 -N “” -f ~/.ssh/id_rsa
   c. クラスター内の別の各ホスト YYY については、root@XXX の SSH 鍵を YYY にコピーします: > ssh-copy-id -i ~/.ssh/id_rsa.pub root@YYY
   d. クラスター内の各ホストに対し、手順 6c を繰り返します。
7. クラスター内の各ホストに対し、手順 6 を繰り返します。

Hadoop のインストールおよび構成

1. すべてのノードに Ambari REPO と鍵をインストールします:
   > wget -O /etc/apt/sources.list.d/ambari.list http://public-repo-1.hortonworks.com/ambari/ubuntu16/2.x/updates/2.5.1.0/ambari.list
   > apt-key adv --recv-keys --keyserver keyserver.ubuntu.com B9733A7A07513CAD
2. パッケージのキャッシュを更新し、アップグレードを適用します:
   > apt-get update –y
   > apt-get upgrade -y
3. 構成ノード上で Ambari サーバーをインストールします:
   > apt-get install -y ambari-server
   > ambari-server setup -j PATH_TO_JAVA_HOME –s
   > ambari-server restart
4. その他のノードに Ambari エージェントをインストールします:
   > apt-get install -y ambari-agent
   > nano /etc/ambari-agent/conf/ambari-agent.ini
   set ‘hostname’ to the IP/FQDN of the ‘config’ node
   > ambari-agent restart
5. 構成ノードで、以下の太字の変数を示された通りに計算または設定します:
   - クラスター内のスレーブ・ノード数:
     NUMBER_OF_SLAVES = X
   - ノード上のメモリー最小容量 (MiB):
     MIN_MEM = 131072
   - OS 用に確保するメモリーの MB (例: 28 GB):
     OS_RESERVE_MB = 31072
   - ノードごとのスレッド数:
     NUM_CORES = num_cpus * cores/cpu * threads/core
   - 許可されるコア数の計算:
     YARN_NUM_CORES = NUM_CORES - 2
   - Yarn の許容メモリー:
     YARN_MEMORY_MB = MIN_MEM - OS_RESERVE_MB = 100000

- Yarn メモリー・アロケーター:
  YARN_MIN_ALLOC_MB = 8192
  YARN_MAX_ALLOC_MB = 32768
  YARN_INCREMENT_MB = 512
- Yarn vcpu アロケーター:
  YARN_MIN_CORES = 1
  YARN_MAX_CORES = 30
  YARN_INCREMENT_CORES = 1
- MapReduce メモリー:
  MAPRED_MAP_MEMORY_MB = 4096
  MAPRED_REDUCE_MEMORY_MB = 8192

a. Ambari でホストを Hadoop の役割へマップする「hostmap」を作成します:

```
> nano ~/ambari-hostmap.json
 {
   “blueprint” : “my_hadoop_cloud”,
   “default_password” : “top-secret”,
   “host_groups” :[
     { “name” : “master1”, “hosts” : [ { “fqdn” : “master1.my_cloud.org” } ] },
     { “name” : “master2”, “hosts” : [ { “fqdn” : “master2.my_cloud.org” } ] },
     { “name” : “master3”, “hosts” : [ { “fqdn” : “master3.my_cloud.org” } ] },
     { “name” : “clients”, “hosts” : [ { “fqdn” : “client.my_cloud.org” } ] },
     { “name” : “slaves”,
       “hosts” : [
         { “fqdn” : “slave1.my_cloud.org” },
         { “fqdn” : “slave2.my_cloud.org” },
         { “fqdn” : “slave3.my_cloud.org” },
         ...
         { “fqdn” : “slaveN.my_cloud.org” }
       ]
     }
   ]
 }
```

b. Create an Ambari blueprint:

```
> nano ~/ambari-blueprint.json
   “configurations” : [
     { “hive-site” : {
         “javax.jdo.option.ConnectionUserName” : “$HIVE_USER”,
         “javax.jdo.option.ConnectionPassword” : “$HIVE_PASS” } },
     { “hdfs-site” : {
         “dfs.datanode.data.dir” : “$DISK_1_MOUNTPOINT, $DISK_2_MOUNTPOINT, ... $DISK_N_
MOUNTPOINT” } } },
     { “yarn-site” : {
         “properties” : {
           “yarn.nodemanager.resource.cpu-vcores” : $YARN_NUM_CORES,
           “yarn.nodemanager.resource.memory-mb” : $YARN_MEMORY_MB,
           “yarn.scheduler.minimum-allocation-mb” : $YARN_MIN_ALLOC_MB,
           “yarn.scheduler.maximum-allocation-mb” : $YARN_MAX_ALLOC_MB,
           “yarn.scheduler.increment-allocation-mb” : $YARN_INCREMENT_MB,
           “yarn.scheduler.minimum-allocation-vcores” : $YARN_MIN_CORES,
           “yarn.scheduler.maximum-allocation-vcores” : $YARN_MAX_CORES,
           “yarn.scheduler.increment-allocation-vcores” : $YARN_INCREMENT_CORES } } },
     { “mapred-site” : {
         “properties” : {
           “mapreduce.map.memory.mb” : $MAPRED_MAP_MEMORY_MB,
           “mapreduce.reduce.memory.mb” : $MAPRED_REDUCE_MEMORY_MB } } }
   ],
   “host_groups” : [
     { “components” : [
         { “name” : “SECONDARY_NAMENODE”},
         { “name” : “HST_AGENT”},
         { “name” : “SLIDER”},
         { “name” : “SPARK_CLIENT”},
         { “name” : “HDFS_CLIENT”},
         { “name” : “ZOOKEEPER_SERVER”},
```

```
            { “name” : “HISTORYSERVER”},
            { “name” : “METRICS_MONITOR”},
            { “name” : “TEZ_CLIENT”},
            { “name” : “APP_TIMELINE_SERVER”},
            { “name” : “RESOURCEMANAGER”}
        ],
        “configurations” : [ ],
        “name” : “master2”,
        “cardinality” : “1” },
    { “components” : [
            { “name” : “SPARK_CLIENT” },
            { “name” : “YARN_CLIENT” },
            { “name” : “HDFS_CLIENT” },
            { “name” : “HST_SERVER” },
            { “name” : “STORM_UI_SERVER” },
            { “name” : “METRICS_MONITOR” },
            { “name” : “INFRA_SOLR_CLIENT” },
            { “name” : “NAMENODE” },
            { “name” : “TEZ_CLIENT” },
            { “name” : “ZEPPELIN_MASTER” },
            { “name” : “NIMBUS” },
            { “name” : “KNOX_GATEWAY” },
            { “name” : “SPARK2_JOBHISTORYSERVER” },
            { “name” : “ACTIVITY_ANALYZER” },
            { “name” : “KAFKA_BROKER” },
            { “name” : “HST_AGENT” },
            { “name” : “MAPREDUCE2_CLIENT” },
            { “name” : “ZOOKEEPER_SERVER” },
            { “name” : “INFRA_SOLR” },
            { “name” : “SPARK_JOBHISTORYSERVER” },
            { “name” : “DRPC_SERVER” },
            { “name” : “METRICS_GRAFANA” }
        ],
        “configurations” : [ ],
        “name” : “master1”,
        “cardinality” : “1” },
    { “components” : [
            {“name” : “MAHOUT” },
            { “name” : “SPARK_CLIENT” },
            { “name” : “YARN_CLIENT” },
            { “name” : “HDFS_CLIENT” },
            { “name” : “SPARK2_CLIENT” },
            { “name” : “METRICS_MONITOR” },
            { “name” : “INFRA_SOLR_CLIENT” },
            { “name” : “TEZ_CLIENT” },
            { “name” : “ZOOKEEPER_CLIENT” },
            { “name” : “HCAT” },
            { “name” : “PIG” },
            { “name” : “HST_AGENT” },
            { “name” : “MAPREDUCE2_CLIENT” },
            { “name” : “SLIDER” },
            { “name” : “HIVE_CLIENT” }
        ],
        “configurations” : [ ],
        “name” : “clients”,
        “cardinality” : “1”
    },
    { “components” : [
            { “name” : “YARN_CLIENT” },
            { “name” : “HDFS_CLIENT” },
            { “name” : “HIVE_SERVER” },
            { “name” : “MYSQL_SERVER” },
            { “name” : “METRICS_MONITOR” },
            { “name” : “HIVE_METASTORE” },
            { “name” : “TEZ_CLIENT” },
            { “name” : “ZOOKEEPER_CLIENT” },
            { “name” : “PIG” },
```

```
                { “name” : “WEBHCAT_SERVER” },
                { “name” : “HST_AGENT” },
                { “name” : “MAPREDUCE2_CLIENT” },
                { “name” : “ZOOKEEPER_SERVER” },
                { “name” : “HIVE_CLIENT” },
                { “name” : “METRICS_COLLECTOR” }
            ],
            “configurations” : [ ],
            “name” : “master3”,
            “cardinality” : “1” },
        { “components” : [
                { “name” : “NODEMANAGER” },
                { “name” : “HST_AGENT” },
                { “name” : “DATANODE” },
                { “name” : “METRICS_MONITOR” },
                { “name” : “SUPERVISOR” }
            ],
            “configurations” : [ ],
            “name” : “slaves”,
            “cardinality” : $NUMBER_OF_SLAVES }
    ],
     “settings” : [
      { “recovery_settings” : [ { “recovery_enabled” : “true” } ] },
      { “service_settings” : [
                { “name” : “HIVE”, “credential_store_enabled” : “true” },
                { “name” : “AMBARI_METRICS”, “recovery_enabled” : “true” }
            ] },
      { “component_settings” : [
                { “name” : “METRICS_COLLECTOR”, “recovery_enabled” : “true” }
            ] }
    ],
     “Blueprints” : {
      “blueprint_name”: “my_hadoop_cloud”,
      “stack_name” : “HDP”,
      “stack_version” : “2.6”
    }
}
```

c. blueprint を登録します:

```
> BLUEPRINT=`cat ~/ambari-hostmap.json`
> AMBARI_SERVER=ambari.my_cloud.org
> BLUEPRINT_NAME=my_hadoop_cloud
> URL=”http://${AMBARI_SERVER}:8080/api/v1/blueprints/${BLUEPRINT_NAME}?validate_topology=false”
> curl -H “X-Requested-By: ambari-script” -d “$BLUEPRINT” -X POST -u admin:admin $URL
```

注: blueprint が正常に登録されたことを確認するために、http://${AMBARI_SERVER}:8080/api/v1/ blueprints/${BLUEPRINT_NAME} に GET CURL 要求を出すこともできます。

d. クラスターを登録します (ホスト・マップ)

```
> HOSTMAP=`cat ~/ambari-blueprint.json`
> AMBARI_SERVER=ambari.my_cloud.org
> CLUSTER_NAME=my_hadoop_cloud
> URL=”http://${AMBARI_SERVER}:8080/api/v1/clusters/${CLUSTER_NAME}?validate_topology=false”
> curl -H “X-Requested-By: ambari-script” -d “$HOSTMAP” -X POST -u admin:admin $URL
```

6. ブラウザーを使って AMBARI_SERVER:8080 までナビゲートし、ログインしてクラスターのデプロイメントが完了するまでオペレーション状況をモニタリングします。

   注: デプロイメントの状況をモニタリングするために、
   http://${AMBARI_SERVER}:8080/api/v1/clusters/${CLUSTER_NAME}/requests/1 に GET CURL 要求を出すこともできます。

HiBench のインストール

クライアント・ノードで、以下の手順を行います:

1. 前提条件をインストールします:
   > apt-get install -y maven git python-numpy libblas-common libblas-dev ¥
   libblas3 liblapack-dev liblapack3 liblapacke ¥
   liblapacke-dev bc
2. HiBench を複製します:
   > mkdir ~/HiBench
   > cd ~/HiBench
   > git clone https://github.com/intel-hadoop/HiBench.git .
3. HiBench を構築します:
   > cd ~/HiBench
   > mvn -Dspark=2.1 -Dscala=2.11 clean package
4. SSH StrictHostKeyChecking を無効にします:
   > nano ~/.ssh/config

   ```
   Host *
       StrictHostKeyChecking no

   [...]
   ```

   注: 手順 4 を行うと、モニタリング/使用状況のレポートを生成できるようになります。

LPCPU のインストール

スレーブ・ノードで、以下の手順を行います:

1. 前提条件をインストールします:
   > apt-get install -y sysstat
2. LPCPU tarball をダウンロードします:
   > wget -o ~/lpcpu.tar.bz2 https://www.ibm.com/developerworks/community/wikis/form/anonymous/ api/wiki/26579cc3-66fe-42b8-baf9-1fcc88445848/page/4a7d2717-05c8-4b78-886e-5e4f9fdf07c1/ attachment/7018590b-7fbe-4852-8d44-79af2d83fffa/media/lpcpu.tar.bz2”
3. LPCPU を解凍します:
   > cd ~
   > tar -xjvf lpcpu.tar.bz2

HiBench の準備

1. 以下の太字の数を計算またはメモします:
   - HDFS のユーザー名
     HDFS_USER = XXX
   - 最初のマスター・ノードの完全修飾ドメイン名
     MASTER_1_FQDN = master1.my_cloud.org
   - すべてのマスター・ノードの完全修飾ドメイン名 (スペースで区切る)
     MASTER_FQDNS = master1.my_cloud.org master2.my_cloud.org master3.my_cloud.org
   - すべてのスレーブ・ノードの完全修飾ドメイン名 (スペースで区切る)
     SLAVE_FQDNS = slave1.my_cloud.org slave2.my_cloud.org slave3.my_cloud.org
   - ワークロードのサイズ (tiny、huge、gigantic、bigdata など)
     HIBENCH_SCALE = gigantic
   - スレーブ数
     S = スレーブ・ノードの合計数
   - スレッド数
     C = スレーブ全体の CPU/スレッドの合計数
   - スレーブ・ノードの最小メモリー (GiB)
     U = 128
   - 占有するメモリーの割合は 0.9 (90 パーセント) が適切です
     W: 0
   - マスター数
     M: 3

- Spark のコア数 (5 が適切です)
  EXECUTOR_CORES = 5
- Spark 実行プログラム数
  EXECUTOR_NUM = floor((C-S) / EXECUTOR_CORES)
- Spark メモリー
  EXECUTOR_MEMORY = floor((U*S) / EXECUTOR_NUM)
- Spark メモリー
  DRIVER_MEMORY = floor((U*S) / EXECUTOR_NUM)
- Spark map の並列処理
  MAP_PARALLELISM = C-S
- Spark reduce の並列処理
  SHUFFLE_PARALLELISM = C-S

2. HiBench Hadoop 構成を設定します:
   > cp ~/HiBench/conf/hadoop.conf.template ~/HiBench/conf/hadoop.conf
   > nano ~/HiBench/conf/hadoop.conf

   [...]

   hibench.hadoop.home /usr/hdp/current/hadoop-client
   hibench.hdfs.master hdfs://${MASTER_1_FQDN}:8020/user/${HDFS_USER}”

   [...]

3. HiBench Spark 構成を設定します: > cp ~/HiBench/conf/spark.conf.template ~/HiBench/conf/spark.conf > nano ~/HiBench/conf/spark.conf

   [...]

hibench.spark.home/	usr/hdp/current/spark2-client
hibench.yarn.executor.num	${EXECUTOR_NUM}
hibench.yarn.executor.cores	${EXECUTOR_CORES}
spark.executor.memory	${EXECUTOR_MEMORY}
spark.driver.memory	${DRIVER_MEMORY}

   [...]

4. HiBench Spark 構成を設定します: > cp ~/HiBench/conf/hibench.conf ~/HiBench/conf/hibench.conf.orig > nano ~/HiBench/conf/hibench.conf

   [...]

hibench.scale.profile	${HIBENCH_SCALE}
hibench.streambench.kafka.home	/usr/hdp/current/kafka-broker hibench.default.map.parallelism ${MAP_PARALLELISM}
hibench.default.shuffle.parallelism	${SHUFFLE_PARALLELISM}
hibench.masters.hostnames	‘${MASTER_FQDNS}’
hibench.slaves.hostnames	‘${SLAVE_FQDNS}’

   [...]

5. ストレージを初期化します:
   - ストレージが既に初期化されている場合は、ストレージをクリアします:
     > sudo -u hdfs hdfs dfs-rmr /HiBench
     > sudo -u hdfs hdfs dfs -rmr /user/${HDFS_USER}
     > sudo -u hdfs hdfs dfs –expunge
     > sudo -u hdfs hdfs dfs -rmr ‘/user/*/.Trash’
   - 必要なディレクトリーを作成します:
     > sudo -u hdfs hdfs dfs -mkdir /HiBench
     > sudo -u hdfs hdfs dfs -chown -R ${HDFS_USER}:hadoop /HiBench
     > sudo -u hdfs hdfs dfs -mkdir /user/${HDFS_USER}
     > sudo -u hdfs hdfs dfs -chown ${HDFS_USER} /user/${HDFS_USER}

6. 準備スクリプトを実行します:
   > cd ~/HiBench
   > ~/HiBench/bin/workloads/ml/kmeans/prepare/prepare.sh

スレーブ・ノードの調整

すべてのスレーブ・ノードで、以下の手順を行います:

1. スワップを無効にします:
   > swapoff -a
2. スワップを再び有効にします:
   > swapon -a
3. ページ・キャッシュをフラッシュします:
   > [-f /proc/sys/vm/drop_caches] && echo 3 > /proc/sys/vm/drop_caches
   注: これはすべてのノードで行うことができますが、今回のテストではスレーブ・ノードのみを対象としました。

LPCPU モニタリングの開始

すべてのスレーブ・ノードで、以下の手順を行います:

1. LPCPU モニタリングを開始します: > cd ~/lpcpu > ./lpcpu.sh duration=99999999 output_dir=output interval=${INTERVAL} dir_name=data
注: テスト期間中は、SSH セッションはオープンにしたまま、LPCPU は実行したままにしておきます (またはバックグラウンドで実行するために start-stop-daemon などでラップします)。

HiBench Kmeans ベンチマークの実行

クライアント・ノードで、以下の手順を行います:

1. ベンチマークを実行します:
   > cd ~/HiBench
   > ./bin/workloads/ml/kmeans/hadoop/run.sh
2. ~/HiBench/report のコンテンツを収集します。

LPCPU モニタリングの停止

すべてのスレーブ・ノードで、以下の手順を行います:

1. SSH セッションの lpcpu.sh を中断するために、 [CTRL] + C を入力します。
2. tarball (output/data.tar.bz2) が作成されるのを待ちます。
3. tarball を解凍し、postprocess.sh スクリプトを実行します。
4. output/data ディレクトリーのコンテンツを収集します。

処理能力の測定

ネットワーク待ち時間やスループット・パフォーマンスを調べるために、TCP ベースの ping とセキュアなコピー・ツールを使用して、3 地域 (米国の東海岸、英国および日本) のうちの 2 つの異なる地域の VM 間について、ネットワーク・パフォーマンスを測定しました。両方のサービスのこれらの面は、paping を使用してテストしました。

対象のクラウド環境上で稼働している Web サーバーに接続し、ping ユーティリティーを使用して TCP-ping テストを実行しました。ポート 80 の 30 秒間の平均待ち時間を測定し、3 回のテスト結果の中央値を採用しました。

scp を使用してファイルのコピー効率をテストし、25MB のファイルと 500MB のファイル、および小さなファイルを含む 100MB のフォルダーの転送所要時間を計測しました。

クラウド・インスタンス構成

それぞれのサービスの 3 つの VM (米国東部、西ヨーロッパ、アジア太平洋 (日本))は以下のように構成しました:

地域

- AWS: バージニア北部、アイルランド、アジア太平洋 (東京)
- IBM Cloud: ワシントン DC、ロンドン、東京

VM インスタンス・タイプ

- すべての VM で Ubuntu 14 を実行しました。

VM 構成

- AWS EC2: m4.xlarge (汎用、4 vCPUs、16GB RAM、EBS ストレージのイメージ)、ネットワーク・パフォーマンス「高」
- IBM Cloud: 4x2.0GHz コア、16GB RAM、1 Gbps パブリック & プライベート・ネットワーク・アップリンク用に構成された計算仮想サーバー

本プロジェクトは、IBM の依頼による実施されました。