

# PENTAHO データ統合

「クラウドやオンプレミス（自社運用型）での  
大量データ処理のスケーリング」

ホワイトペーパーの提供:



<http://www.bayontechnologies.com>

Nicholas Goodman

[ngoodman@bayontechnologies](mailto:ngoodman@bayontechnologies)

ホワイトペーパーの翻訳:



株式会社KSKソリューションズ（Pentaho日本代理店）

<http://www.pentaho-partner.jp>

# イントロダクション

Pentahoデータ統合 (PDI: Pentaho Data Integration) 3.2は高度なクラスタリング・パーティショニング機能を提供します。それによって、高価な単一ノードマシンで「スケールアップ (1台のサーバーでの機能向上)」する代わりにETL (Extract Transform Load) デプロイメントを「スケールアウト (サーバー数を増やして機能向上)」することが可能になります。データ量は常に増えていくため、バッチウィンドウ (バッチ処理時間) はどのようなデータウェアハウスでも問題となります。

PDIスケールアウト特性はプロプライエタリなソリューションと比べ、費用対効果がよいことに加え、リスクを減少させることができます。この仕様書では、PDI (Pentahoデータ統合) のスケールアウトの性能や最大40のノードにわたって何十億ものレコードを処理する能力についてのレビューをします。またこの仕様書は、クラウド中でデータを処理するためのパフォーマンスメトリクスや価格の比較の説明をしつつ、「オンデマンド」でPDIのクラスターを構築するためのインフラとしてのアマゾンエラスティックコンピューティングクラウド (EC2: Amazon Elastic Compute Cloud) について説明していきます。

この仕様書は以下の問いに答えます:

## **問1: クラスターにノードを追加するとき、PDIは比例して変化しますか?**

1つのノードで処理をするのに4時間かかる場合、4つのノードでそれを行うと1時間で終わるのでしょうか。バッチ処理時間を減少させるために単純にクラスターにコンピューターノード加えることができるかどうかを検証していきたいと思います。バッチ時間要件を満たすために適切な数のノードをデプロイできるため、直線的なスケーラビリティがあるのが理想的です。

## **問2: より大量のデータを処理するとき、PDIは比例して変化しますか?**

処理しているデータセットのサイズを倍にすると、PDIはデータ処理するのに2倍時間がかかるのでしょうか。PDIクラスターが増量されたデータをそのまま扱うことができるかどうかを検証していきます。

## **問3: クラウドでのETLに関する主要な価格とパフォーマンス指標は何ですか?**

私たちのクラウドの実験から、いくつかの総合的な性能やコストを見ていきます。最終的にはインハウスでの開発のシナリオと比較をすることになりますが、ここでは十億のレコードのソート・集計について、いくらコストがかかるのかといったより具体的な指標を求めています。

**問4: アマゾンEC2にPDIをデプロイした人は何をしておく必要があるのでしょうか。**

クラウドでPDIクラスターをデプロイする上でのベストプラクティスや、一連のよくある問題に対処することが必要です。そのための参考情報を提供します。

これらの4つの質問をまとめた結果と分析を提供することに加えて、実験からの特定のテストケース、データセット、およびEC2で実行された結果について詳細に説明するいくつかのセクションがあります。PDIデータ変換またはテストデータセットジェネレーターはここで提示されたテストケースと結果を見るためにダウンロードすることができます。（付録 Bを見てください—ダウンロードリンクを参照）

## **Pentahoデータ統合（PDI）クラスター**

PDIクラスターはデータ変換のパフォーマンスとスループットを増強するために構築されるものです。特にそれらは、並行してデータセットを処理する従来からの“divide and conquer”を実行するために構築されています。PDIにはバージョン2.4以降クラスタリング機能があり、クラスタリング新機能がリリースごとに追加されています。PDI 3.2に搭載された新しいクラスター機能は、ダイナミックに利用可能なサーバーのクラスターを作成する能力を改善し、ランタイム時にPDIクラスターを管理するのに必要な仕事量をかなり減少させます。これらの「ダイナミッククラスター」機能は管理をずっと単純、簡単にし、この仕様書で議論するクラウドのデプロイメントを可能にします。

PDIクラスターには、強力なマスター/スレーブトポロジー（接続形態）があります。クラスター内のマスターは一つですが、スレーブはたくさんあります。データ変換は、実行時にマスターとスレーブに分割され、そしてクラスター上のすべてのサーバーにデプロイされます。クラスターにおける各サーバーは、「Carte（カルテ）」と呼ばれる動作中のアプリケーションです。Carte（カルテ）は、データ変換を受ける、実行する、モニターするための非常に軽量なWebサービスインターフェースを実行する小さなPDIサーバーです。

しかしながら、PDIクラスターは、「高可用性」や「ジョブ管理」システムとしてフォ

ールトトレランスやロードバランスを扱うことを意図していません。PDIクラスタリングとパーティショニングの構造はランタイム時において比較的単純です。PDIクラスターはクラスターにおいて、ノードのCPU/ロードまたは実測性能に基づく最適化を一切行いません。

PDIクラスタリングはソフトウェアの機能です。いかなる特定のクラウドベンダーAPI/実装にも依存しません。それは容易にハードウェアにデプロイできます。

## Amazon EC2

EC2はサーバー、ブロックデバイス、IPアドレスなど、計算リソースのダイナミックな提供を可能にします。このインフラは企業が計算を必要としているときマシンを提供し、必要なくなればこれらのリソースの解放を可能にします。サービスのレートは使用する分だけ契約し、自らが使用する分の代価を払うだけです。1時間100台のマシンを使用する価格は100時間1台のマシンを使用するのと同じです。

未加工の計算リソース（サーバー）の提供に加え、EC2はEBS（Elastic Block Storage）と呼ばれるブロックデバイスを提供します。

このサービスは、EC2のためのストレージ・エリア・ネットワーク（SAN）のように動作します。ブロックデバイスがEC2サーバーにマウントされ、ファイルシステムが作成されて使用することができます。

長期のハードウェアコミットメントなしに使用したものに対してのみ支払うので、これは短期間にたくさんの計算リソースを使用する人にとっては非常に魅力的なモデルとなります。これらのおおよそのコストを検討していきます。：

- 40個の小さなサーバー（ひとつのバーチャルコアに対してメモリ1.7GB）を一時間使用する= 4.00ドル
- 20TBのブロックストレージを1時間使用する= 2.80ドル

## ETLとクラウド：最良の組み合わせ

計算リソースのプロファイルと毎晩、毎週、毎月とのETLバッチプロセスの間には、理想的な組み合わせが必要となります。通常日々のバッチETL処理は夜中に行われ、前日から決められたデータを処理します。バッチウィンドウ（バッチ処理時間）はプロセス（処理するために利用可能なデータ）の始まりからデータ処理が終了する（レポートがユーザーに対して準備ができています）時までの時間の総計です。バッチウィンドウは比

較的決まった時間を維持してはいます。（午前12時01分に処理し始めて、そのレポートを午前8時にレポートを見ることができます。）しかし、多くの組織では相当な速度で処理しなければならないほどデータ量が増えています。

クラウドコンピューティングの魅力的なコストとPDIのETLの処理のパフォーマンスは、クラウドのデプロイメントをより理想的なものにします。

# 実験

## テストデータ : TPC-H

実験ではトランザクション処理性能評議会 (Transaction Processing Council) からの TPC-Hデータセットを使用することにしました。このデータセットを選んだ理由は以下の通りです。

- TPCはデータの異なったデータのスケールを生成するデータジェネレータを提供している。実験では、3つのデータセットを生成させました。スケール50、100、300 (GB)。
- TPC-Hは比較的よく知られているDSS (Decision Support System) データセットである。投資時期や組み込みテストデータセットに時間と努力を注ぎ込まず、比較的知名度の高いData Warehouseデータベースベンチマークのためのデータセットを使うことができました。
- テストデータセットはこれらのテストから独立している。このデータセットはこれらのテストで良い結果を出すためにつくられたものでないことを納得していただくとおもいます。

付録Bをご覧ください - 私たちが実験に使用したデータファイルの詳細を含むTPC-Hドキュメントへのリンクのリファレンスがあります。また、これらのドキュメントは (ER図のような) ファイル間の関係を示す図を含んでいます。

“lineitem (ラインアイテム)”ファイルは、主なトランザクション・レコードであり、TPC-Hデータセットの中で最も大きいものです。それには小売店で購入されたアイテムの情報が含まれています。私たちの実験ではいくつかのリレーションシップが使用されています。

- lineitem (ラインアイテム) は「Part / Supplier」データファイルに対して party/supplierという二重バリューの外部キーを持っています。
- supplier (供給者) はcountry of origin (原産国) に外部キーを持っています。ファイルはパイプ(|)によって区切られる様々なデータ型 (キャラクタ、整数、およびナンバー) がある “フラット” ファイル (flat file) です。実験に使用されるデータファイルはTPCが提供する “dbgen” プログラムを使用して生成した後は、いかなる変更もされていません。

ファイル	フィールド数	スケール	サイズ (GB)	行数
lineitem.tbl	16	50	36.82	300,005,811
lineitem.tbl	16	100	74.12	600,037,902
lineitem.tbl	16	300	225.14	1,799,989,091

supplier.tbl	7	50	0.07	500,000
supplier.tbl	7	100	0.13	1,000,000
supplier.tbl	7	300	0.40	3,000,000
nation.tbl	4	すべて	0.00	24

注意: この実験は決して公式TPC-Hベンチマークではありません。TPC-Hベンチマークは、それらをデータベースで実行するのに特に重要な要件です。この実験はこれらのいずれも行いません。

### ETLシナリオ

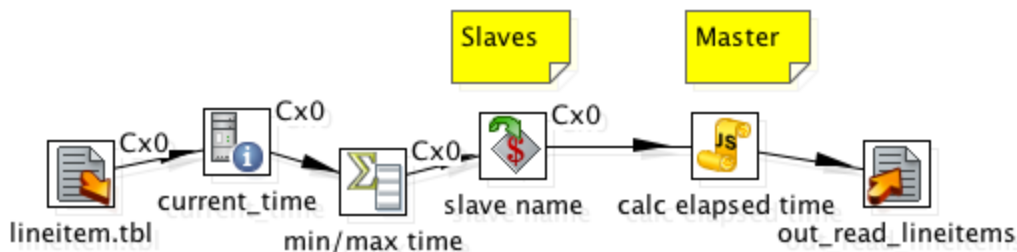
私たちの実験での間に答えるために、以下のようなETLが必要です。

- 比較的単純で、PDIに詳しくない読者でもETL処理全体を理解できる。
- 何らかの重要な処理をする。
- 比較的一般に使用されるケースに適合する。

いくつかの異なったETLシナリオが評価され、最終的に2つのETLデータ変換でテストしました。いくつかハイレベルなデータ変換の説明が付録2のフルスクリーンショットと共にここで提示されています。

### READ (read\_lineitems\_lazy.ktr)

READは、単純にクラスターがどれくらい速くファイル全体を読むことができるかを見るベースラインです。レコードを読み込んだら、それらのレコードには追加の処理をしません。READはデータ変換のスレーブ部分を終わらせて、次にこの特定のスレーブサーバーがどう働いたかの情報を返します。



READは実際に使用することはないでしょう。単にそのインフラを評価して、大きいデータファイルの読み込み (reading) のための基礎データを得るのに使用します。

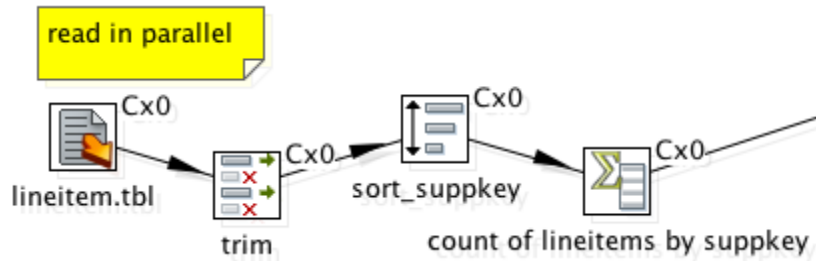
## READアウトプットファイル (out\_read\_lineitems)

READはサーバー (out\_read\_lineitems) に関するサーバー上の個々のノードパフォーマンスに関して、いくつか役に立つ情報をアウトプットします。そこにはサーバー名、処理されるレコード番号、そしてサーバーあたりのスループット（単に時間あたりの処理能力）が含まれています。

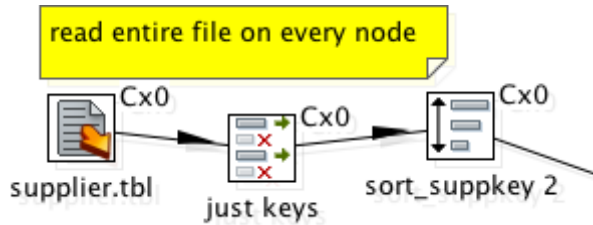
スレーブ	レコード数	経過時間 (秒)	スループット/ノード
ダイナミックスレーブ [10. 250. 15. 69:8080]	30144428	894	33718. 60
ダイナミックスレーブ [10. 251. 199. 67:8080]	29916897	901	33204. 10
ダイナミックスレーブ [10. 250. 19. 102:8080]	29917123	903	33130. 81
ダイナミックスレーブ [10. 250. 47. 5:8080]	30144739	908	33199. 05
ダイナミックスレーブ [10. 251. 42. 162:8080]	29988406	920	32596. 09
ダイナミックスレーブ [10. 251. 71. 159:8080]	29916947	942	31758. 97
ダイナミックスレーブ [10. 251. 122. 49:8080]	29915685	949	31523. 38
ダイナミックスレーブ [10. 250. 11. 95:8080]	29916158	950	31490. 69
ダイナミックスレーブ [10. 251. 203. 114:8080]	29916164	1044	28655. 33
ダイナミックスレーブ [10. 250. 137. 204:8080]	30229264	1095	27606. 63

## SORT (sort\_lineitems\_group\_by\_nationkey. ktr)

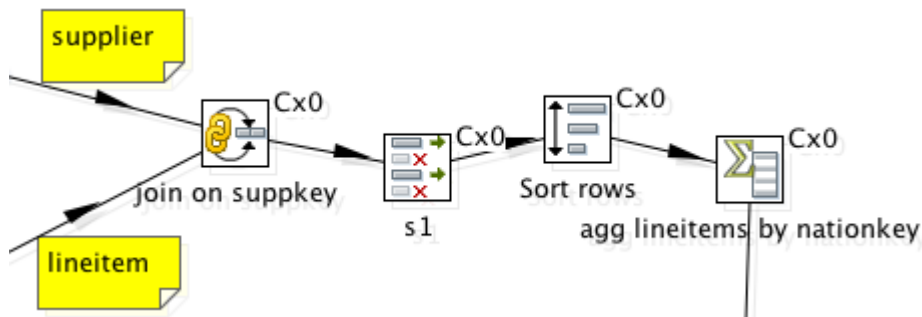
SORTは大きなトランザクションファイルのためのソートや結合、集計などを表しています。SORTはデータファイルを読み込むという点でREADと似ていますが、それをまとめて「サプライヤー毎のラインアイテムの数 (COUNT OF LINEITEMS by SUPPLIER)」を構築するためにファイルをソートし、集計します。



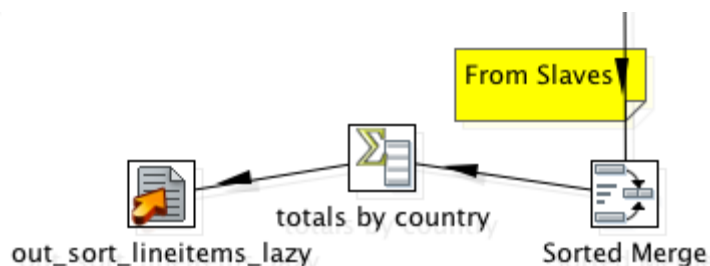
SORTはまた別のREADや、別の大きなファイル（supplier）のSORTをします。この場合、集計はありませんが、データは「サプライヤー毎のラインアイテムの数（COUNT OF LINEITEMS by SUPPLIER）」分岐にJOINするためにSORTされます。



SORTして、両方のファイルから共有“suppkey”バリューを使用する別の大きいファイル（supplier）とJOINします。「COUNT OF LINEITEMS by SUPPLIER」をすべての供給者（supplier）レコードに合わせた後、nationkeyでソート、集計します。



結果（統合された総計を構築すること）をマスターサーバーに集めて、マスターサーバー上のデータファイルに結果をアウトプットします。



### アウトプットファイルのSORT (out\_sort\_lineitems\_group\_by\_nationkey)

アウトプットは供給国ごとのラインアイテム数のサマリーです。シンプルな25行のファイルでラインアイテムデータファイル内のレコードの総数となります。例えば、TPC-H スケール300ファイルに関して、アウトプットファイルは以下のようになります。

国別キー (nationkey)	ラインアイテム (Lineitem) のカウント
0	71886458
1	71969681
2	72038468
3	72179079
4	72209741
5	72146529
6	72178370
7	72055521
8	72184094
9	71870036
10	71974851
11	72051132
12	71627623
13	71529955
14	71795817
15	72079318
16	71897393
17	71862655
18	71974391
19	72037468
20	72201477
21	72056815

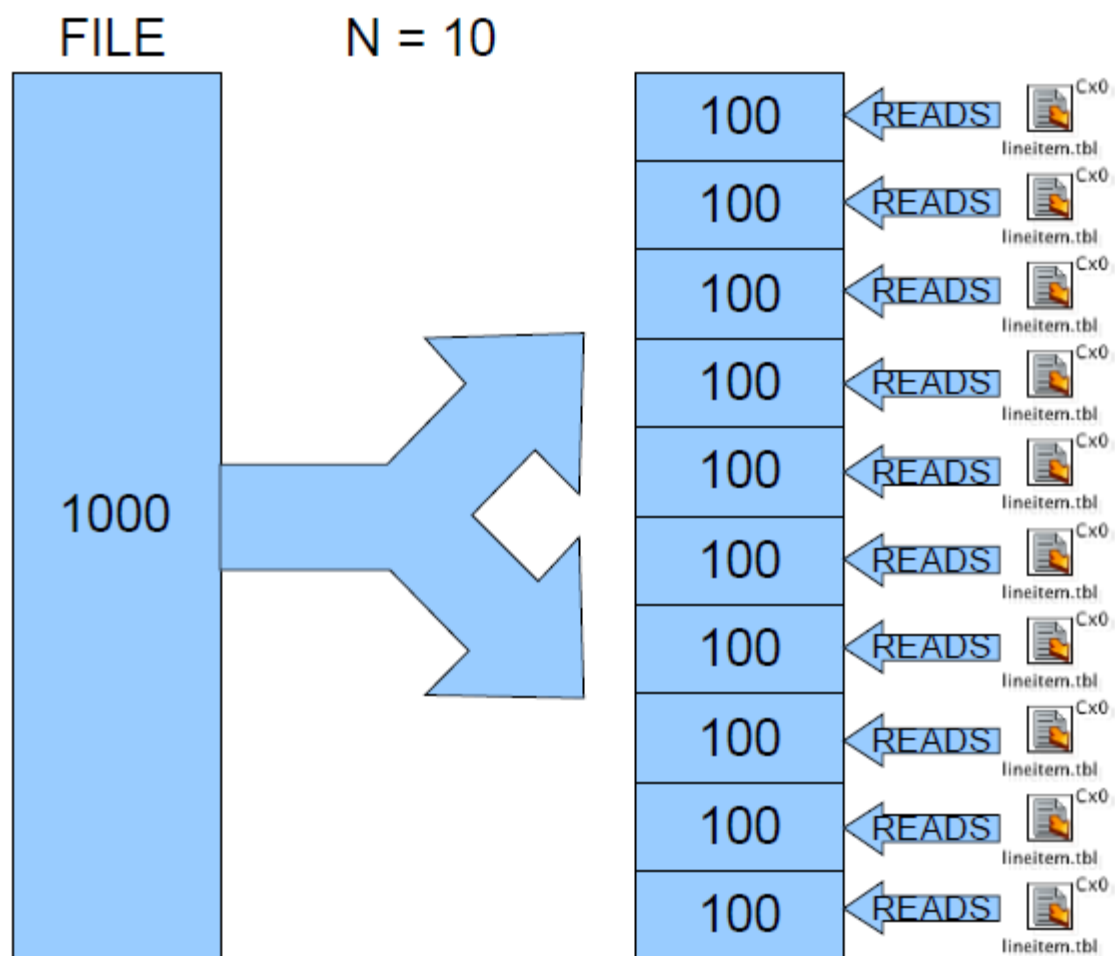
22	71873005
23	72432871
24	71876343

各テストで、私たちは正しい結果を得られたことを証明しました。これは、（READとSORTの両方での）ローのカウンと（SORTの）結果のチェックを含んでいます。

### パラレルファイル処理

Cx0という名称は、このステップがクラスターで実行されることを示します。デフォルトで、ほとんどのステップは全く追加クラスター設定する必要がありません。慎重を要する一連のタスク（計算、デコード、フィルタリングなど）のほとんどは独立して働き、特別なクラスター設定は必要ありません。しかし、インプットとアウトプットは、クラスターで動作する場合、データの一部を調整するために設定する必要があります。

CSV インプットステップは、クラスターでデプロイされる場合、ファイルの「セグメント」のみを読み込む（READ）「並行で実行」という設定パラメータがあります。これは、各ノードがファイルの約1/N部分を読み込む（READ）のを可能にします。



またこれは、「並行で実行」というのが、それぞれがファイルの異なるセクションを処理するために、すべてのノードが同じファイルへREADアクセスを必要とすることを意味しています。

## EC2クラスター

### インスタンスイメージ (AMI)

EC2は3つのインスタンスサイズを提供します。: スモール (small)、ラージ (large)、Xラージ (XLarge) です。スモール (small) は32ビットのイメージで、ラージ (large)、Xラージ (XLarge) は64ビットのイメージです。この実験では、私たちは以下のハードウェア仕様でスモールインスタンスを使用しました:

- メモリ1.7 GB
- 1つのEC2計算ユニット (1つのEC2計算ユニットにつき1つのバーチャルコア)
- 160 GB インスタンスストレージ、32ビットのプラットフォーム

私たちはFedora 8で、自身のマシンサーバーイメージ (AMI) を構築しました。このマ

シンイメージは、標準Amazon EC2ビルドツールを使用しながら構築しました。マシンは、「user data」をスタートアップ時に受け入れるように設定されました。AMIイメージはこの実験の間、スタティックな状態でしたが、サーバーのセットを始動したとき様々なスレーブサーバー設定バリューを提供しました。サーバーの設定を開始したとき、私たちは、以下の設定可能なパラメータをパスしました。

#### □ pdi-distribution-url

どこでPDIを入手するかについてのURL。PDIのアップグレードバージョンは、AMIを変更する必要がないので、AMIはダウンロードして、ダイナミックにPDIをインストールします。

例: <http://www.bayontechnologies.com/pdi-ce-3.2.0-RC1-r10482.zip>

#### □ carte-slave-server-xml

これはAMIが「Carte (カルテ)」を開始するXMLです。MASTERに関して、これは特殊要素を一切含んでいません。スレーブサーバーのために、スレーブサーバーがマスターに自身を登録できるようにマスターのホスト名を含む必要があります。

例:<slave\_config> <slaveserver>

<name>carte-slave</name><hostname>localhost</hostname>

<network\_interface>eth0</network\_interface> <!--OPTIONAL --> <port>8080</port>

<username>cluster</username><password>cluster</password> <master>N</master>

</slaveserver> </slave\_config>

#### □ data-volume-device

このインスタンスに取り付けられたEBSブロックデバイスのデバイスロケーション。

例:/dev/sdl

#### □ data-volume-mount-point

EBSブロックデバイスを取り付けるロケーション。

例:/data

#### □ java-memory-options

Carte (カルテ) 開始のためのメモリオプション。

-Xmx1024m

この「user data」はサーバーを開始するための要求の一部としてEC2にパスされます。

## EBSボリューム

EBSは、同じファイルの部分を読み込む能力をクラスター中のすべてのノードに提供するメカニズムです。EBSボリュームは、一度に、1つのサーバーにのみ取り付けことができ、1~1000GBのサイズで定めることができます。

私たちは1000GB EBSボリュームを作成して、XFSファイルシステムをそれにインストールしました。XFSは、ext3の代わりに大きいファイル（2.4GB以上）サポートを可能にするのに使用されます。私たちは、3つ異なるスケールのTPC-Hデータをこれと同じボリューム上に生成しました。そのため、どのボリュームでも実行できます。

一度に1つのEBSボリュームは1つのサーバーにしか取り付けることができないので、クローンを作って、実行中のインスタンスに対して個々のボリュームを実装する必要があります。私たちは、EBSスナップショット機能を使用してボリュームのスナップショットを取りました。EBSはまたスナップショットから新しいボリュームの素早く作成するので、データセットを持っているNノードそれぞれに対して1つのボリューム（Nボリューム）を作ることができます。

## スクリプト

私たちは、PDI EC2クラスターのライフサイクルの管理を助けるためのスクリプトを作りました。これには以下の主要な機能が含まれます。

- ・ マスター/スレーブインスタンスの作成
- ・ TPC-Hデータセットスナップショットからマスタスレーブボリュームの作成
- ・ インスタンスへのボリュームの追加
- ・ すべてのインスタンスでのステータスレポート

□クラスターでのデータ変換の実行

NマシンのPDIクラスターPDIクラスターは以下のプロセスを使用して作成されます：

- ・ マスターボリュームをスナップショット（ec2-create-volume）から作成する。
- ・ Nボリュームをスナップショット（ec2-create-volume）から作成する。
- ・ マスター・スレーブ（ec2-run-instance）を開始する。
- ・ いったんマスターが実行した後、ボリューム（ec2-attach-volume）を取り付ける。
- ・ Nスレーブを開始する（ec2-run-instance）
- ・ NボリュームをNスレーブに取り付ける（ec2-attach-volume）。
- ・ スレーブが自動的にマスターに登録される。

PDIクラスターは以下のプロセスを使用して破壊されます：（A PDI cluster was DESTROYED using the following process）

- スレーブインスタンスを終了する。(ec2-terminate-instances)
- スレーブボリュームを取り外す。(ec2-detach-volume)
- マスターインスタンスを終了する。(ec2-terminate-instances)
- マスターボリュームを取り外す。(ec2-detach-volume)
- スレーブボリュームを削除する。(ec2-delete-volume)
- マスターボリュームを削除する。(ec2-delete-volume)

ライフサイクルスクリプトを使用することで、10台、20台、40台のクラスターマシンが、数分間で作成・破壊されました。

*注意: スクリプトは個人のセキュリティー証明書を含んでいます。それらは `thepdi_scale_out_benchmark_kit.zip`には含まれていません。*

## 結果

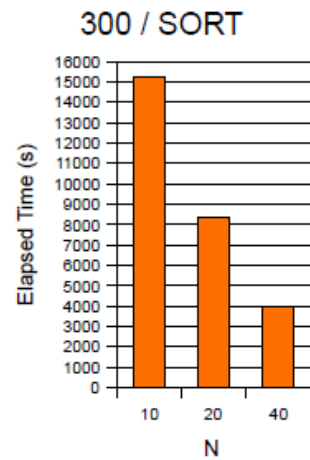
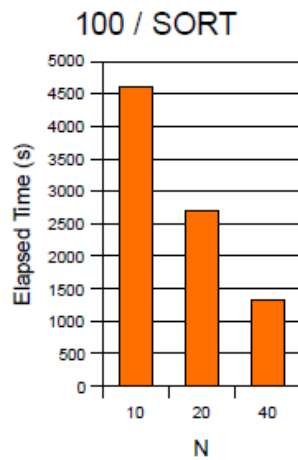
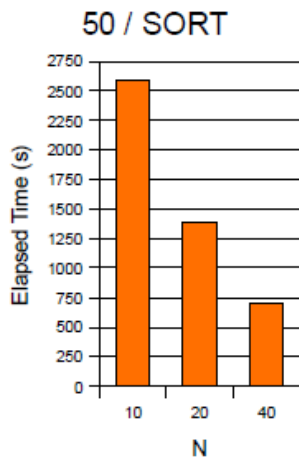
私たちは、SORTとREADの両方で3つのスケール（50、100、300）のために3つの異なるクラスターサイズ（10、20、40）を使用し実験を行いました。結果は以下の表にまとめられます。

スケール	行数	ノード (N)	経過時間	ロー/秒	変換
50	300005811	10	1095	273978	READ
50	300005811	20	836	358859	READ
50	300005811	40	640	468759	READ
100	600037902	10	1787	335779	READ
100	600037902	20	1736	345644	READ
100	600037902	40	1158	518167	READ
300	1799989091	10	6150	292681	READ
300	1799989091	20	5460	329668	READ
300	1799989091	40	3835	469358	READ
50	300005811	10	2588	115922	SORT
50	300005811	20	1389	215987	SORT
50	300005811	40	704	426145	SORT
100	600037902	10	4615	130019	SORT
100	600037902	20	2680	223895	SORT
100	600037902	40	1318	455264	SORT
300	1799989091	10	15252	118017	SORT
300	1799989091	20	8390	214540	SORT
300	1799989091	40	4014	448428	SORT

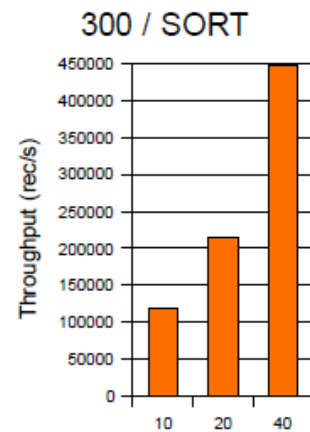
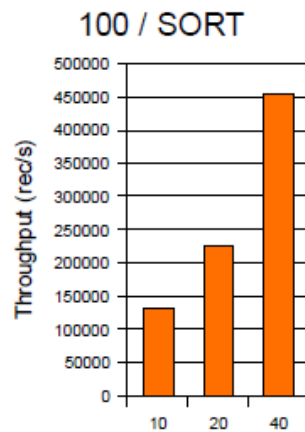
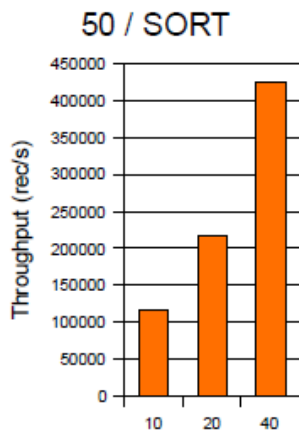
実験の完全な結果はベンチマークキットのエクセルファイルに含まれており、ここで全体は提示されていません。私たちの実験の中でのテスト結果を調べる予定です。

**問 1：追加ノードをクラスターに加えるとき、PDIは比例して変化しますか？**

**SORT：経過時間**



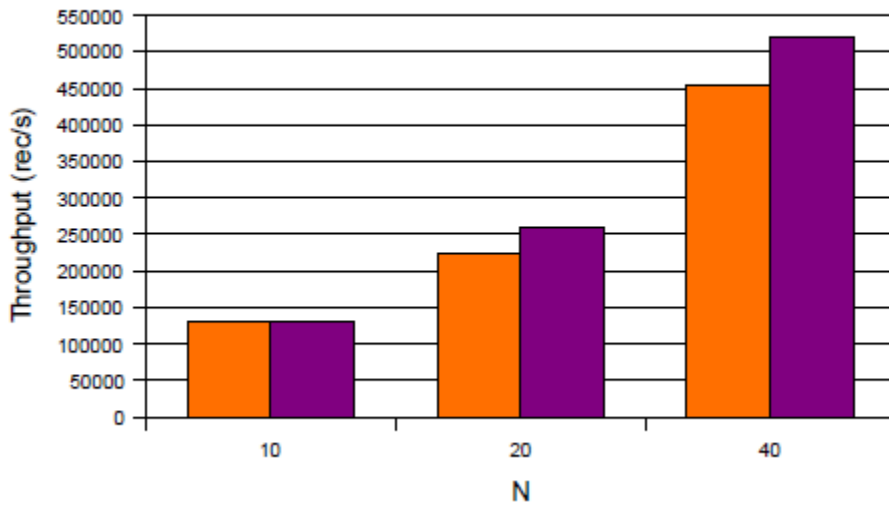
SORT: スループット (レコード/秒)



SORTからの結果は、より多くのノードをクラスターに加えるとスループットと総合的な処理が比例に近づくという仮説はあっています。ただし確実なわけではありません。

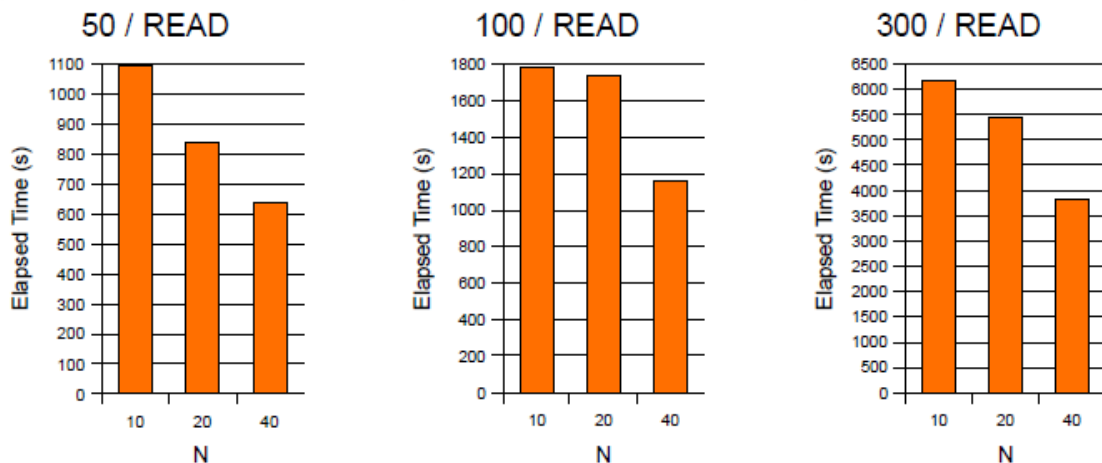
3つのクラスターサイズにわたって100 / SORTを考えてみましょう。N=10のベースラインから始めるなら、20や40で予測していた完全な比例を達成するにはどのくらい (N 10 \* 2, や N 10 \* 4) をとる必要があるのでしょうか。

### 300 / SORT

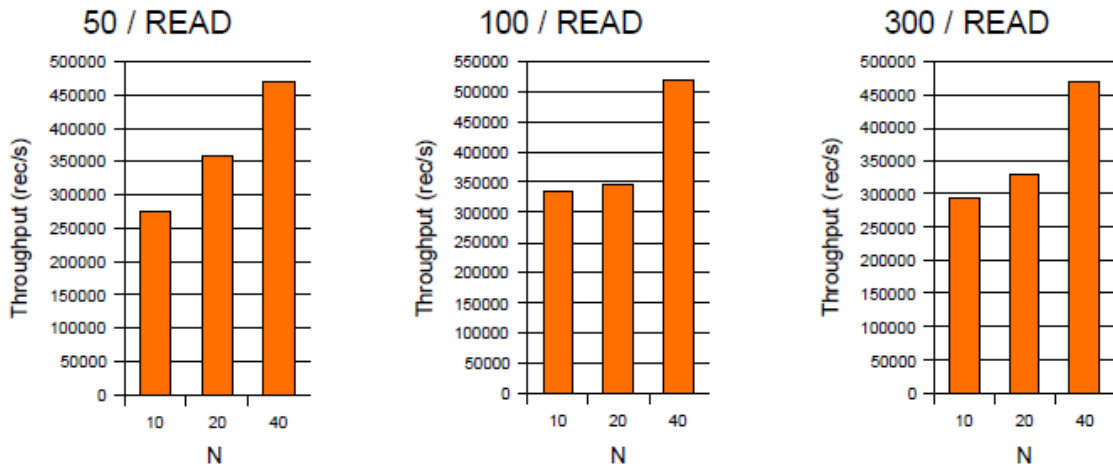


パフォーマンスは、私たちの予想に完全には適合しません。そしてもっとノードを加えるといくぶん減退を見せ初めます。このいくつかは、クラウドコンピューティングリソースのバリエーション（質問4の結果で議論しました）に関係がありそうです。例えば、300SORT/N40は実験の間、予想されたN10\*4ナンバーよりも速かったことは、実験中潜在的なハードウェア条件の変化があったことを示しています。しかしながら、データはより多くのノードを加えるとき必ずしも比例関係でないことを示しています。

### READ: 経過時間



### READ: スループット (レコード/秒)



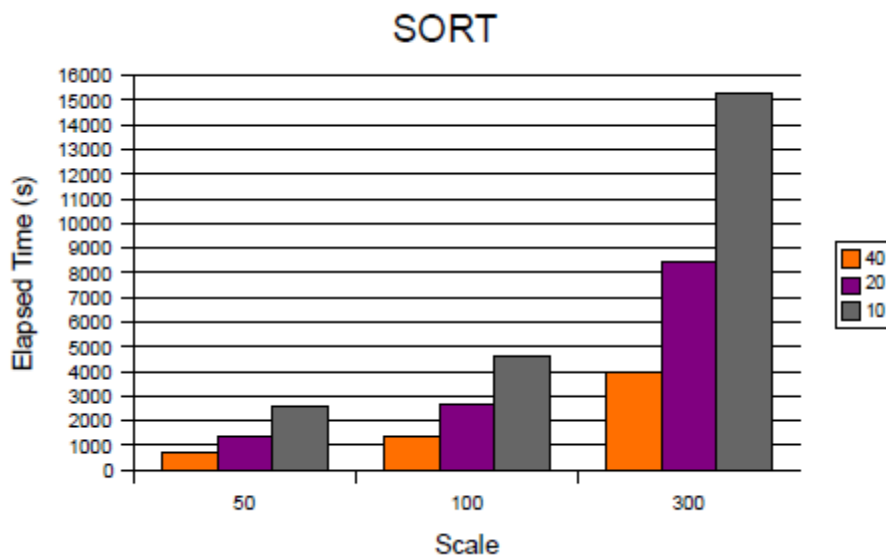
N40の性能は予想されたN10\*4には及びませんでした。実際、N40はN10の性能の2倍にすら及んでいません。

READデータ変換ではノード間のコミュニケーションはなく、すべてのノードはファイルの異なるセクションで独自に動作しています。実際の処理は全くなく、少量のデータ(数KB)だけがスレーブからマスターへ通過します。このことは私たちの基本的なストレージシステム(EBS)の性能が同じスナップショットに基づくボリュームに対するREAD性能が直線的にスケールアップされていることを示します。残念ながら、EBS上のRAW性能のために取り付けられたボリュームで、あらゆる種類のOSレベルベンチマークを実行することまではできませんでした。

**回答:** 私たちの実験は100%制御された環境ではなく、潜在的なクラウドのインフラが結果を歪曲したと思います。たとえそうだとすると、私たちのSORT(I/Oを少し阻止する)は比例の変化に近いものを示し、新しいノードを加えるときのパフォーマンスを大いに改善します。EC2上のPDIはほとんど比例して変化しますが、EBS変化がおそらく原因となり確実に比例するわけではありません。

**問2:** もっと大量のデータを処理するとき、PDIは比例して変化しますか?

異なったスケールとノードのための結果は既に提示されました。スケールを(50、100、300)と増加させてどのように比例していくか、SORTのパフォーマンスを見ていきましょう。



すべてのノードサイズ100におけるバリューはおおよそ50\*2です。すべてのノードサイズ300におけるバリューはおおよそ50\*6です。私たちは、リニアよりも良い結果を観察することができました。いくつかのバリューは、リソース/EBSパフォーマンスにおいて注目すべき変化を示しました。

回答: PDIはデータ量が増えるに従ってリニアにスケールすると言えます。

**問3: クラウドでのETLの主要な価格とパフォーマンスメトリクスは何ですか?**

最初に、私たちのクラスタの何らかの総合的な機能指標に触れましょう。私たちは、3つのスケールすべてで各ノードの総合的なSORT性能を見ていきました。そして平均して各ノードにつき「1秒あたり11373レコード」であることがわかりました。

比較的可線形なスケールを示しているデータ変換でのSORTのパフォーマンスの平均です。このことは異なるスケール（データサイズ）、クラスタサイズ（クラスタ中のインスタンス）で比較的、確実にこの数字のもとで使用できることを示しています。

ノード	スループット	スループット/ノード
40	443278.79	11081.97
20	218140.49	10907.02
10	121319.17	12131.92
平均SORTスループット/ノード		11373.64

これは、「スモール」サイズのE02インスタンスのロードコストです:

EC2費用	費用/時間
サーバー時間	\$0.10
EBSボリューム (1000 GB)	\$0.14
S3からのEBSロード	\$0.01
Bandwidth (帯域幅)	\$0.00
合計	\$0.25

1つのノードの1時間につきいくつのローをSORTできるのでしょうか？

$11373 \times 60 \text{ (秒)} \times 60 \text{ (分)} = 40942800$  : ロー/時間/スモールノード

では、10億行のローをクラウドで処理するときの価格はどのくらいでしょう？

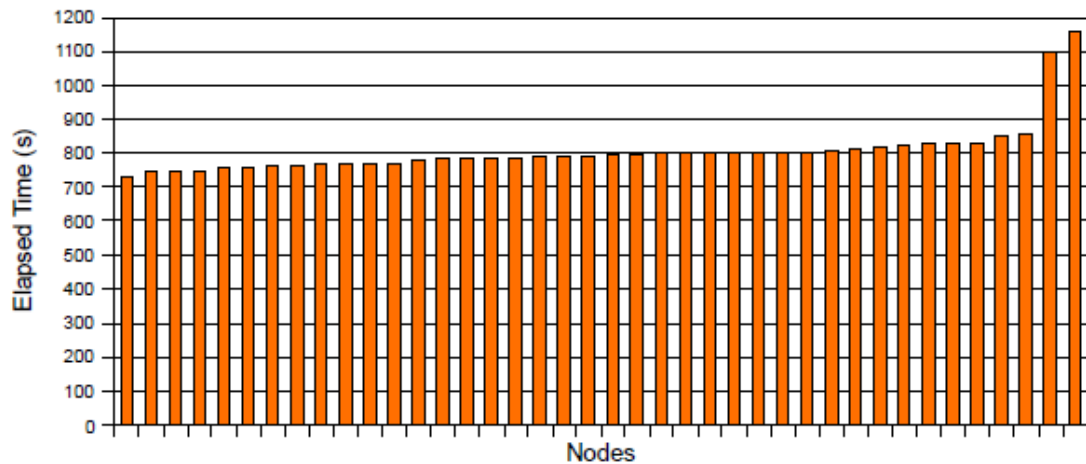
$(\$0.24 \times 1000000000) / 40942800 = \$5.86$  : EC2で10億行のローをSORTするためのコスト

*注意: これらの計算はクラスター始動時間 (40ノードクラスターで約20分) と、ほんのわずかなbandwidthコストを考慮していません。*

**問4: アマゾンEC2にPDIをデプロイした人は何をしておく必要があるのでしょうか。**

READデータ変換は個々のノード性能指標を作り出しました。私たちは個々のノード性能のいくつかの重要な変化を観察していきました。個々の性能の違いがインスタンス (同じ物理サーバー上の他の仮想マシンが原因のもの) か、それともそれがEBSへの接続 (ファイルの読み込み) なのかはわかりませんが、それらが同じサイズ/数のローを処理していたとしても、いくつかのノードが他のものより最大58%長くかかっているのを確認しています。

## 100 / READ / 40 Nodes



大部分は比較的スムーズで、ノード1-38ではそれぞれ100秒以内に完了します。ノード39と40は、相対的なデータ変換のパフォーマンスが「最も遅い」ノードに相当するため、他に比べてかなり長い時間がかかってしまいます。全テスト結果は、ノード単位のさらに正確なパフォーマンス数を図るスループットに基づく平均を示しています。しかし実際の使用においてはさほど関係ありません。PDIクラスターはそれらの最も遅いノードと同じくらいの速さです。

PDIは計算リソースの均質なセットを仮定しています。それはファイルセグメントの分割であり、ノードに対する配分はマシンのパフォーマンスやサイジングのバリエーションは考慮していません。

MapReduce / Hadoopのようなシステムは、これを収容するように最適化して組み込まれます。そして、負荷に応じて、異なるサーバーにひとまとまりの仕事を再割り当てします。PDIにはそのような機能はありません。

### EBSボリュームパフォーマンス

READパフォーマンスに基づいて、より多くのノードがスナップショットから作成されるボリュームから読み込んでいる場合、EBSパフォーマンスが下がることを確認しました。これはおそらく後のEBSが非同期的にS3からのスナップショットをロードすることが原因で、それらがロードされる前でブロックにヒットします。それぞれのEBSボリュームは、ファイルの異なる部分にヒットして、その結果、異なるEBS/S3ブロックにヒットします。私たちは、EBSをバイパスし、S3ディレクトリからの平行なREADパフォーマンスの

さらなる実験を行う必要があります。

## プライベートIP対パブリックIP

アマゾンEC2には、各マシンにつき2つのIPアドレスがあります： プライベートIPとパブリックIP。プライベートIPは、アマゾンEC2ネットワーク内部での通信に用いられ、2つのネットワークでより速いとされています。パブリックIPは、外の世界との通信に使用されます。

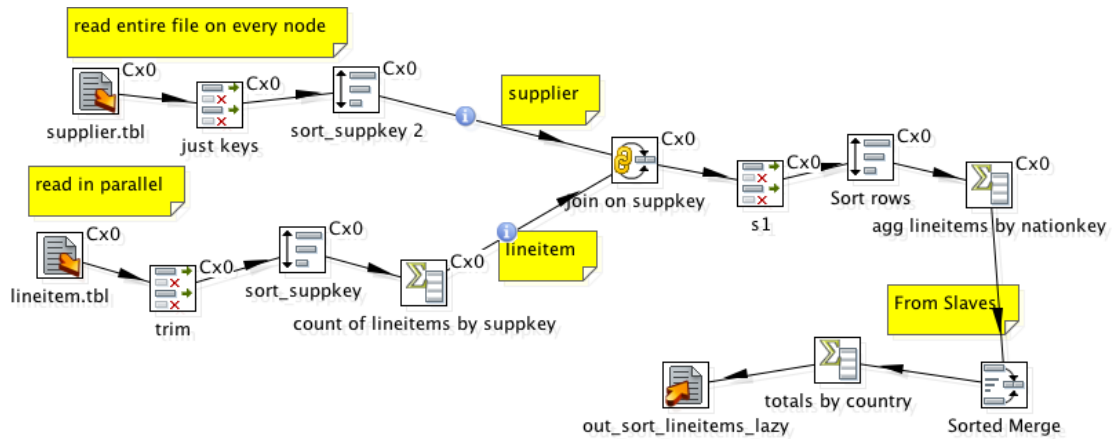
私たちのクラスタースタートアップ/モニタリングスクリプトでは、私たちはパブリックIPをサーバーとの通信に使用する必要がありますが、私たちはサーバーに相互プライベートIPで通信してほしいと思っています。より速いことに加えて、Amazon EC2ではたとえEC2ネットワーク内であってもパブリックIPを使用する際チャージが存在する一方で、プライベートIPにはbandwidthチャージは全くありません。これによって、私たちは直接リモートサーバーへのPDIビジュアルデザイナーを通してクラスタリングしているデータ変換を実行できないようになってしまいました。私たちは、ラッパージョブを作成して、それをマスターに送り、コマンドラインからそれをそこへ実行させました。PDI3.2には、ジョブをエクスポート、そして送る機能がありますが、私たちがこれらの実験を実行したとき、それはコマンドライン“kitchen. sh”では利用できませんでした。

経費を節約しネットワークパフォーマンスを最大化するためにもノード間のコミュニケーションではプライベートIPアドレスを使用するようPDIを設定するようご注意ください。

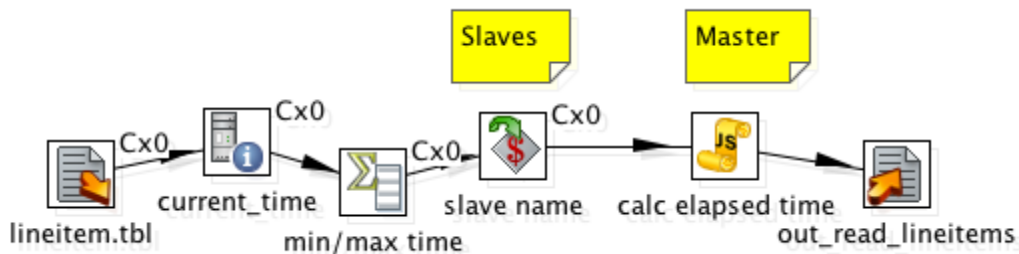
# 付録

## 付録A テストデータ変換

sort\_lineitems\_group\_by\_nationkey.ktr is available in  
pdi\_scale\_out\_benchmark\_kit.zip



read\_lineitems\_lazy.ktrはpdi\_scale\_out\_benchmark\_kit.zipで利用することができます。



## 付録B リファレンス

Benchmark Kit

[http://www.bayontechnologies.com/bt/ourwork/pdi\\_scale\\_out\\_benchmark\\_kit.zip](http://www.bayontechnologies.com/bt/ourwork/pdi_scale_out_benchmark_kit.zip)

TPC-H

<http://www.tpc.org/tpch/default.asp>

TPC-H Data generator

[http://www.tpc.org/tpch/spec/tpch\\_2\\_8\\_0.tar.gz](http://www.tpc.org/tpch/spec/tpch_2_8_0.tar.gz)

TPC-H Document

<http://www.tpc.org/tpch/spec/tpch2.8.0.pdf>

Amazon EC2

<http://aws.amazon.com/ec2/>

EBS

<http://aws.amazon.com/ebs/>

PDI

<http://kettle.pentaho.org/>

Dynamic Clusters

<http://wiki.pentaho.com/display/EAI/Dynamic+clusters>