Amazon Redshift

AWS Black Belt Tech Webinar 2016

アマゾンウェブサービスジャパン株式会社
ソリューションアーキテクト 相澤恵奏

2016.07.20

amazon web services

自己紹介

名前：相澤　恵奏（あいざわ　けいぞう）

所属：

- アマゾンデータサービスジャパン株式会社
- エコシステムソリューション部
- パートナーソリューションアーキテクト

好きなAWSサービス:Redshift, RDS

AWS Black Belt Tech Webinar 2016

- AWSJのTechメンバがAWSのプロダクトを深堀りして解説するWebセミナー
  - サービスの概要、使いどころの説明
  - アップデートのキャッチアップ

- **毎週水曜 18～19時**
- 申し込みサイト
  - http://aws.amazon.com/jp/event_schedule/
- Twitter ハッシュタグ
  - **#awsblackbelt** で確認

内容についての注意点

- **本資料では2016年7月20日時点のサービス内容および価格についてご説明しています。最新の情報はAWS公式ウェブサイト(http://aws.amazon.com)にてご確認ください。**

- **資料作成には十分注意しておりますが、資料内の価格とAWS公式ウェブサイト記載の価格に相違があった場合、AWS公式ウェブサイトの価格を優先とさせていただきます。**

- **価格は税抜表記となっています。日本居住者のお客様が東京リージョンを使用する場合、別途消費税をご請求させていただきます。**

AWS does not offer binding price quotes. AWS pricing is publicly available and is subject to change in accordance with the AWS Customer Agreement available at http://aws.amazon.com/agreement/. Any pricing information included in this document is provided only as an estimate of usage charges for AWS services based on certain information that you have provided. Monthly charges will be based on your actual use of AWS services, and may vary from the estimates provided.

Agenda

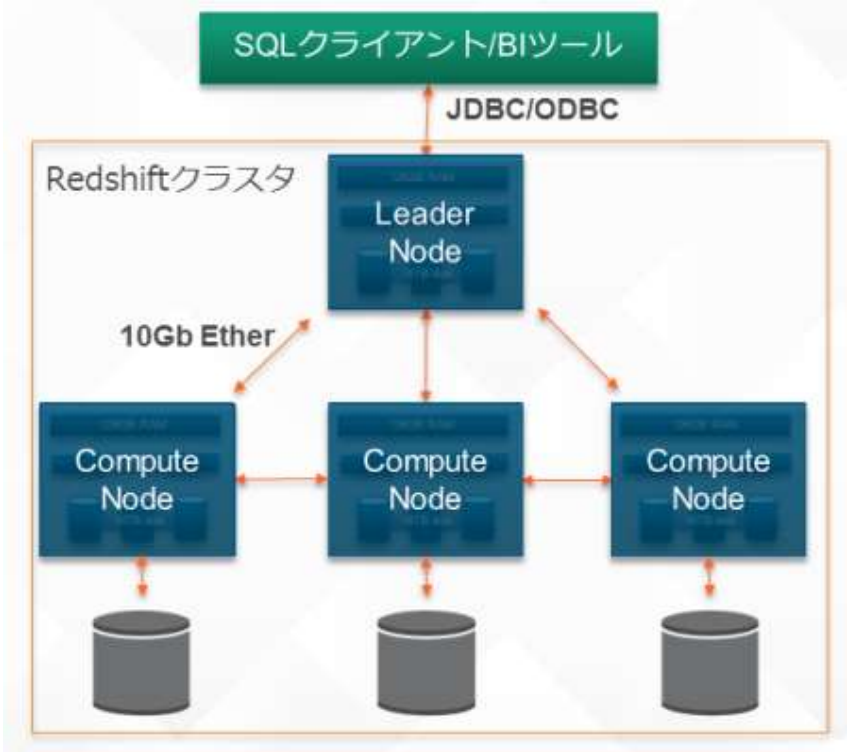
- Amazon Redshiftとは？
- パフォーマンスを意識した表設計
- Amazon Redshiftの運用
- Workload Management (WLM)
- パフォーマンスチューニングテクニック Top 10
- まとめ

Agenda

- **Amazon Redshiftとは？**
- パフォーマンスを意識した表設計
- Amazon Redshiftの運用
- Workload Management (WLM)
- パフォーマンスチューニングテクニック Top 10
- まとめ

Amazon Redshiftの概要

- クラウド上のDWH
  - 数クリックで起動
  - 使った分だけの支払い
- 高いパフォーマンス
  - ハイ・スケーラビリティ
- 高い汎用性
  - PostgreSQL互換のSQL
  - 多くのBIツールがサポート

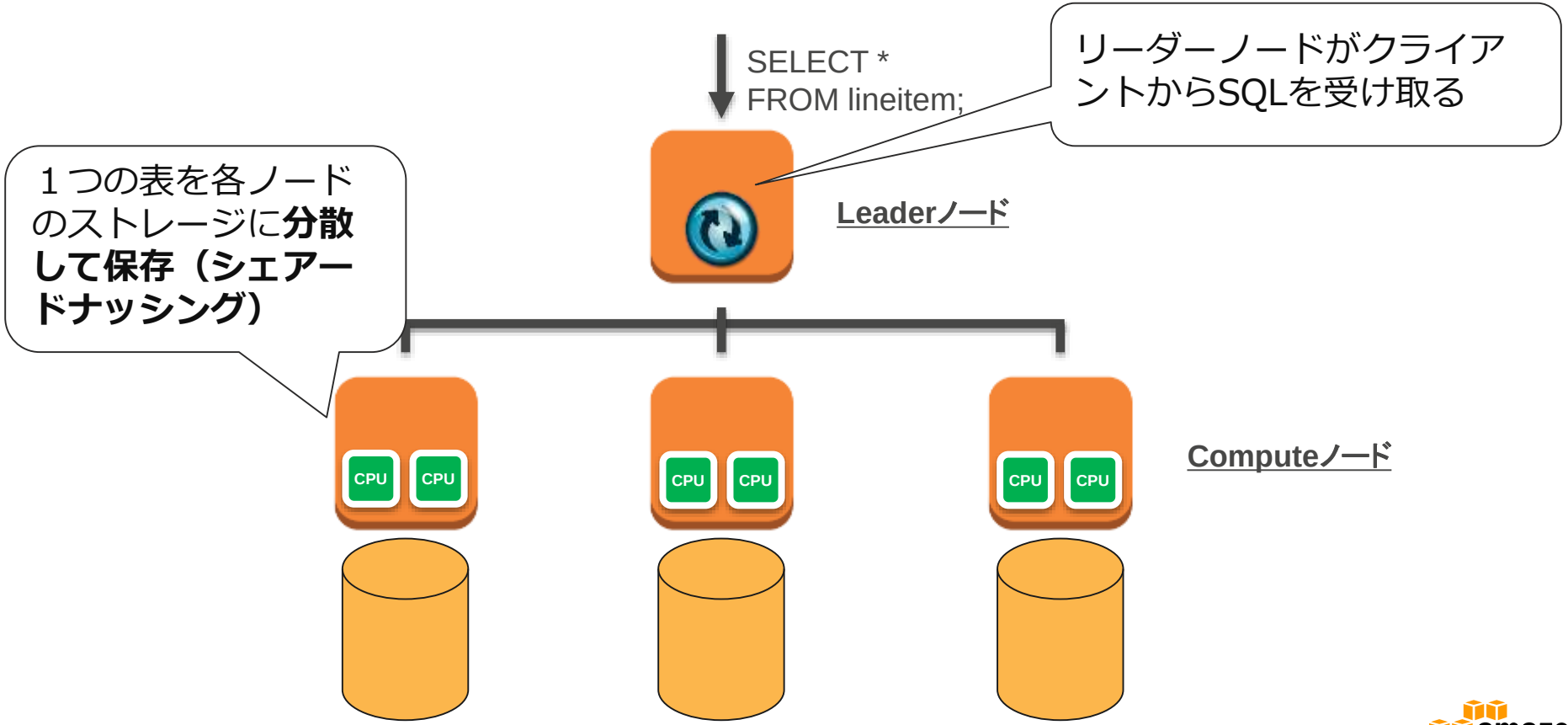


MPPとシェアードナッシングがスケールアウトの鍵

- **MPP：Massive Parallel Processing**
  - １つのタスクを複数のノードで分散して実行する仕組み
  - Redshiftではリーダーノードがタスクをコンピュートノードに分散して実行する
  - ノードを追加する（スケールアウト）でパフォーマンス向上可能

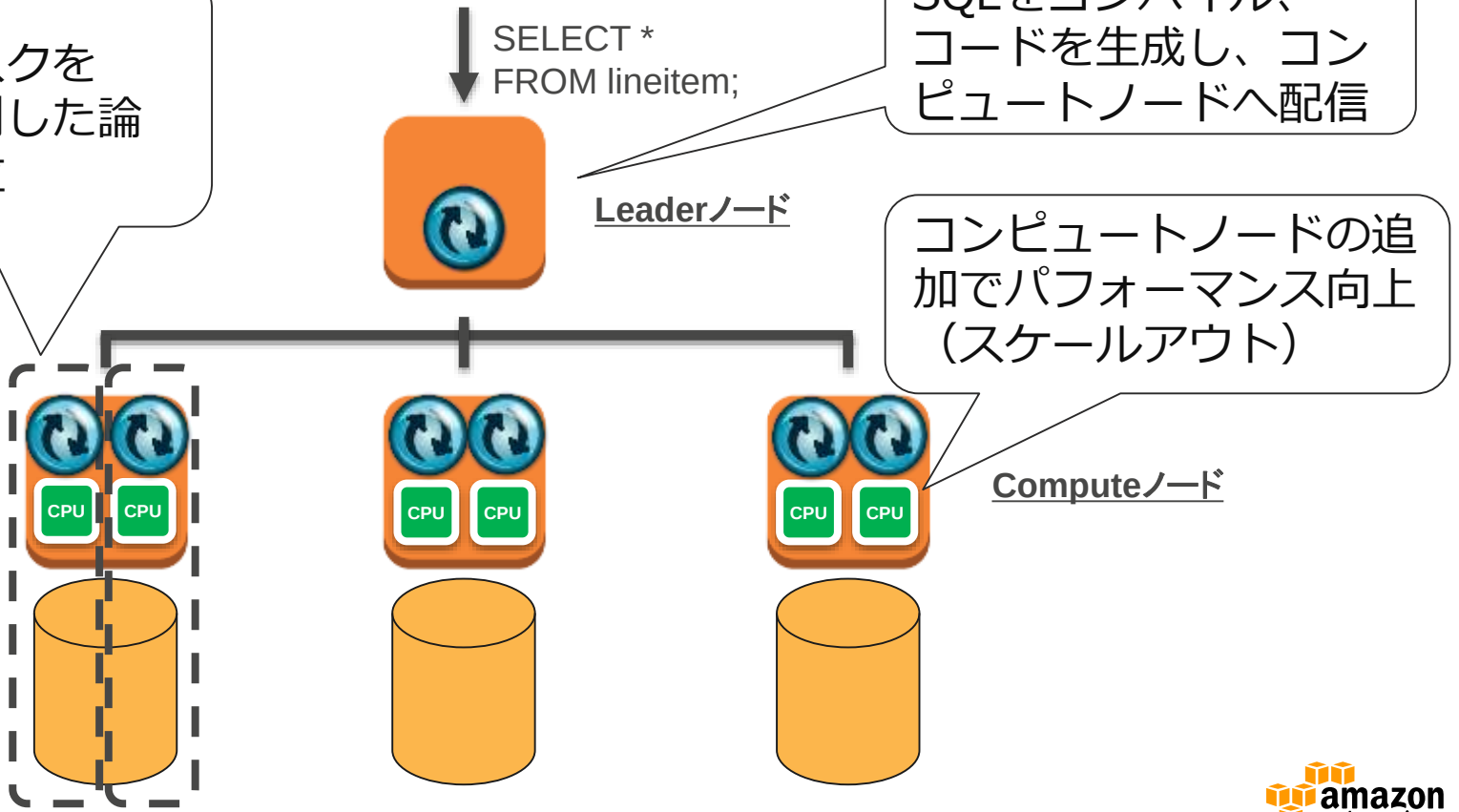
- **シェアードナッシング**
  - ディスクをノードで共有**しない**構成
  - ディスクを共有するとノード数が増えた時にボトルネックになるため、それを回避
  - ノードとディスクがセットで増えていく

Redshiftの構成①



Redshiftの構成②

スライス＝
メモリとディスクを
ノード内で分割した論
理的な処理単位



amazon web services

ノードタイプ

- SSDベースのDCとHDDベースのDSから選択
  - データは圧縮されて格納されるため、ストレージ総量より多くのデータが格納可能
- 最大128ノード：2.0PByteまで拡張可能
  - ノードタイプと数は後から変更可能

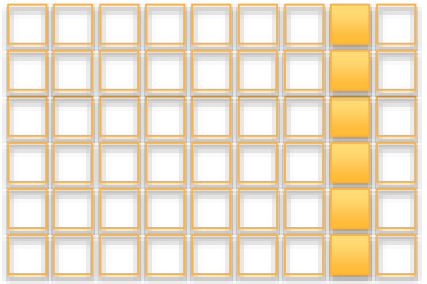
※価格は東京リージョンにおいて2016年7月20日時点のものです

DC1 - Dense Compute					
	vCPU	メモリ(GB)	ストレージ	ノード数	価格(※)
dc1.large	2	15	0.16TB SSD	1~32	$0.314 /1時間
dc1.8xlarge	32	244	2.56TB SSD	2~128	$6.095 /1時間
DS2 – Dense Storage					
ds2.xlarge	4	31	2TB HDD	1~32	$1.190 /1時間
ds2.8xlarge	36	244	16TB HDD	2~128	$9.520 /1時間

【補足】リーダーノードと利用費用

- リーダーノードもコンピュートノードも同じノードタイプで構成される
- リーダーノード分は利用費用が不要
- １ノード構成にした場合、リーダーノードとコンピュートノードが１ノードに同居する

IOを削減する① - 列指向型（カラムナ）



DWH用途に適した格納方法

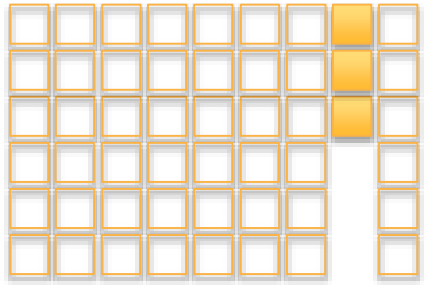
・行指向型（他RDBMS）

orderid	name	price
1	Book	100
2	Pen	50
	…	
n	Eraser	70

・列指向型（Redshift）

orderid	name	price
1	Book	100
2	Pen	50
	…	
n	Eraser	70

IOを削減する② - 圧縮

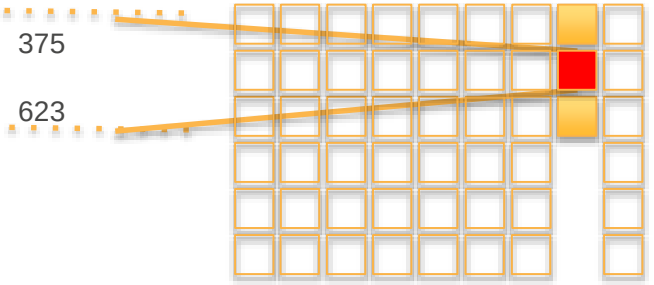


- データは圧縮してストレージに格納される
- カラムナのため類似したデータが集まり、高い圧縮率
- エンコード（圧縮アルゴリズム）は列ごとに選択可能
- COPYコマンドやANALYZEコマンドで圧縮アルゴリズムの推奨を得ることが可能

```
analyze compression listing;

  Table   |     Column      | Encoding
----------+-----------------+----------
 listing | listid          | delta
 listing | sellerid        | delta32k
 listing | eventid         | delta32k
 listing | dateid          | bytedict
 listing | numtickets      | bytedict
 listing | priceperticket  | delta32k
 listing | totalprice      | mostly32
 listing | listtime        | raw
```

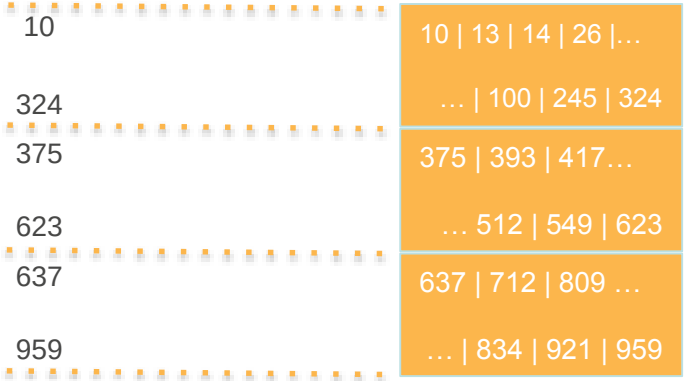
IOを削減する③ - ゾーンマップ



Redshiftは「ブロック」単位で
ディスクにデータを格納
１ブロック＝１MB

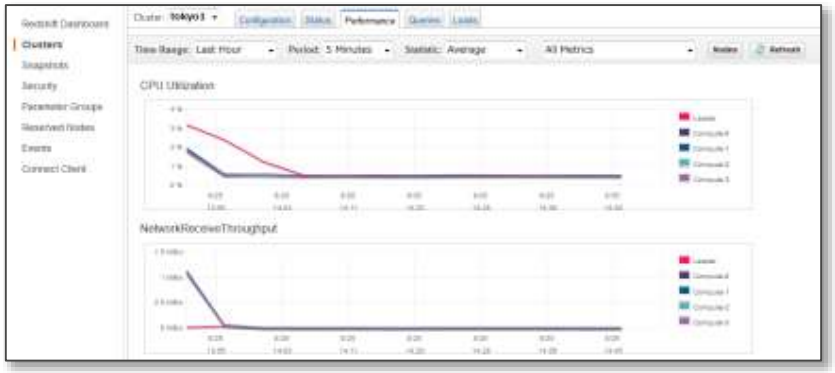
ブロック内の最小値と最大値をメモリに保存

不要なブロックを読み飛ばすことが可能



amazon web services

フルマネージドサービス



設計・構築・運用の手間を削減

- 数クリックで起動
- １時間単位の費用
- ノード数やタイプは後から変更可能
- バックアップ(Snapshot)やモニタリング機能を内蔵
  - GUI（マネジメントコンソール）
  - API経由で操作も可能
- パッチ適用も自動的
  - メンテナンスウィンドウでパッチの時間帯を指定可能

Redshiftが向く用途

- 特化型のデータベースのため、適した用途に使うことでパフォーマンスを発揮します

- Redshiftに向くワークロード
  - 巨大なデータ・セット（数百GB～ペタバイト）
  - １つ１つのSQLが複雑だが、同時実行SQLは少ない
  - データの更新は一括導入

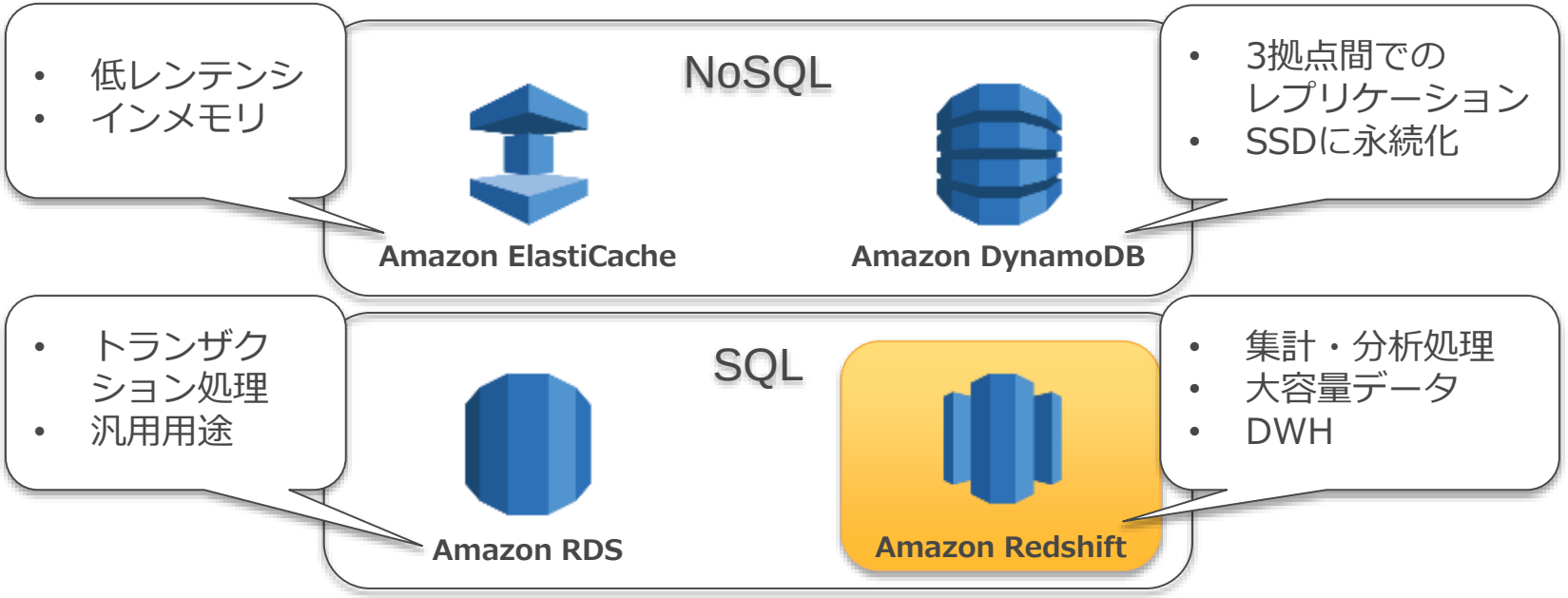
- ユースケース
  - データウェアハウス（DWH）
  - ユーザがクエリーを作成する（自由クエリー）（BI等）

Redshiftの特徴を生かせないユースケース

- SQLの並列実行数が多い（※同時接続数ではなく同時実行数）
  - RDS（MySQL ,PostgreSQL, Oracle, SQL Server)を検討
- 極めて短いレーテンシが必要なケース
  - ElastiCache (インメモリDB)やRDSを検討
- ランダム、かつパラレルな更新アクセス
  - RDSもしくはDynamoDB (NoSQL)を検討
- 巨大なデータを格納するが集計等はしない
  - DynamoDBや大きいインスタンスのRDSを検討

Amazon Redshiftの位置づけ

データ・ストアの特性に応じた使い分け



amazon web services

Agenda

- Amazon Redshiftとは？
- **パフォーマンスを意識した表設計**
- Amazon Redshiftの運用
- Workload Management (WLM)
- パフォーマンスチューニングテクニック Top 10
- まとめ

DDLによるパフォーマンスの最適化

- ディスクIOを削減する
  - サイズを減らす
  - 読む範囲を減らす
- ノード間通信を削減する
  - 通信しないようなデータ配置

ディスクIOを削減する：型を適切に選択する

- 型を適切に選択してサイズを節約する
  - 不必要に大きい型を選択しない
  - BIGINT(8バイト)よりも、INT(4バイト)やSMALLINT(2バイト)
  - FLOAT(8バイト)よりも、REAL(4バイト)
  - 日付は文字列(CHAR)で格納せずTIME型を使用

Redshiftで利用可能な型

- 下表の型をサポート
- charはシングルバイトのみサポート
- varcharはUTF-8形式でのマルチバイトをサポート

データ型	別名	説明
SMALLINT	INT2	符号付き 2 バイト整数
INTEGER	INT、INT4	符号付き 4 バイト整数
BIGINT	INT8	符号付き 8 バイト整数
DECIMAL	NUMERIC	精度の選択が可能な真数
REAL	FLOAT4	単精度浮動小数点数
DOUBLE PRECISION	FLOAT8, FLOAT	倍精度浮動小数点数
BOOLEAN	BOOL	論理ブール演算型（true/false）
CHAR	CHARACTERNCHAR、BPCHAR	固定長のキャラクタ文字列
VARCHAR	CHARACTER VARYING、NVARCHAR、TEXT があります。	ユーザーによって定義された制限を持つ可変長キャラクタ文字列
DATE		カレンダー日付（年、月、日）
TIMESTAMP	TIMESTAMP WITHOUT TIME ZONE	日付と時刻（タイムゾーンなし）

参照）
http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/c_unsupported-postgresql-datatypes.html

ディスクIOを削減する：適切な圧縮方法の選択

- 圧縮を行うことで、一度のディスクアクセスで読み込めるデータ量が多くなり、速度の向上が見込める

- 圧縮のエンコード（アルゴリズム）が複数用意されており、CREATE TABLEで各列に選択することが可能

```
CREATE TABLE table_name (
   列名 型 ENCODE エンコード,
)
```

- 動的には変更できない (作りなおして INSERT … SELECT)

圧縮エンコーディングの種類

- データの特性に応じたエンコーディングを選択するのが理想
- ANALYZE COMPRESSIONコマンドで推奨を確認可能
  - 先にデータの投入が必要
- LZOは比較的多くのケースで有効

Encoding	対応型	
RAW	ALL	圧縮無し
BYTEDICT	BOOL以外	列値で辞書を作る。値の種類が少ないときに効果的
DELTA/DELTA32K	数値	連続する数値を差分で表現して圧縮
LZO	varchar向け	長い文字列の圧縮に有効
MOSTLY(8|16|32)	数値	数値型に対し実際の数値範囲が小さい場合に有効
RUNLENGTH	ALL	列がソートされている等、同じ値が繰り返されるパターン
TEXT255/32K	文字列	同じ単語が頻出する場合

圧縮エンコーディングの確認

- pg_table_def のencoding列で確認可能

```
mydb=# select "column",type,encoding from pg_table_def where tablename='customer_enc';
    column    |         type          | encoding
--------------+-----------------------+----------
 c_custkey    | integer               | delta
 c_name       | character varying(25) | lzo
 c_address    | character varying(25) | lzo
 c_city       | character varying(10) | bytedict
 c_nation     | character varying(15) | bytedict
 c_region     | character varying(12) | bytedict
 c_phone      | character varying(15) | lzo
 c_mktsegment | character varying(10) | bytedict
```

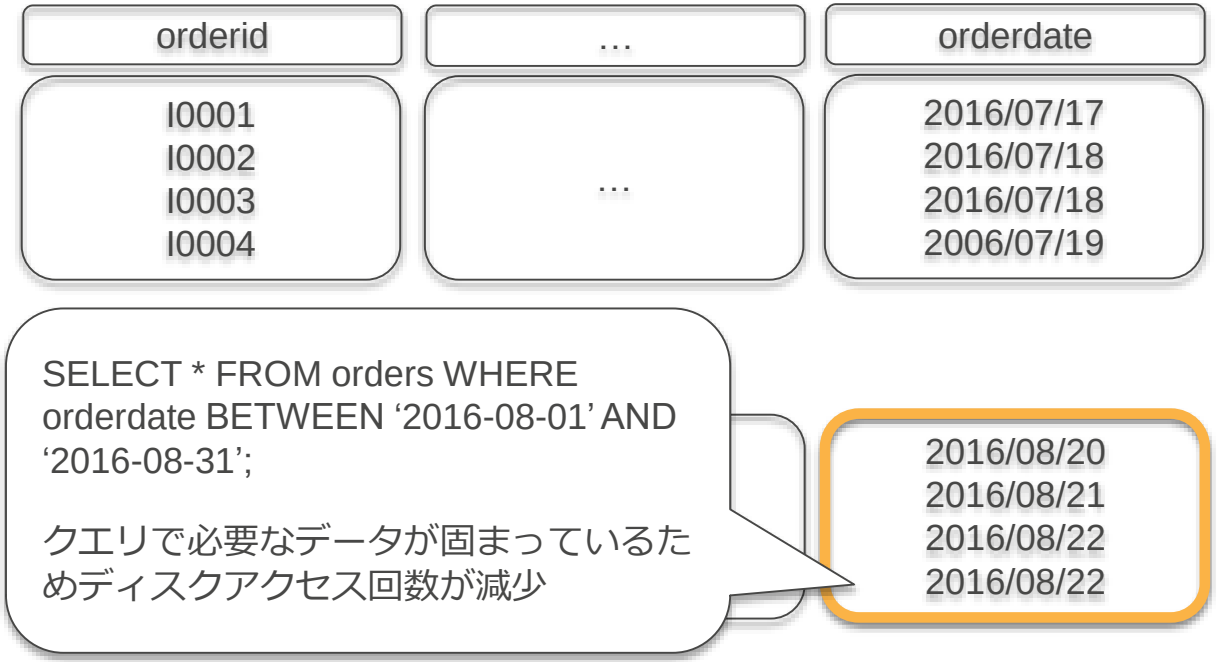
ディスクアクセスの範囲を最小にする

- SORTKEY
  - SORTKEYに応じて、ディスク上にデータが順序を守って格納
  - クエリー・オプティマイザはソート順序を考慮し、最適なプランを構築
  - CREATE TABLE時に指定。複数列が指定可能
    - CREATE TABLE t1(…) SORTKEY (c1,c2 …)

- SORTKEYの使いどころ
  - 頻繁に特定のカラムに対して、**範囲または等式検索を行う**場合
    - 例）時刻列
  - 頻繁にジョインを行う場合、該当カラムをSORTKEYおよびDISTKEYとして指定→ ハッシュ・ジョインの代わりにソート・マージ・ジョインが選択される

SORTKEY の例

- orderdate 列をSORTKEY に指定した場合：



Interleaved Sort Key

- 新しいSort keyのメカニズム
- 最大８つまでのSort Key列を指定でき、それぞれ同等に扱われる

```
CREATE TABLE ~
…
INTERLEAVED SORTKEY (deptid, locid);
```

- 旧来のSortで複数のキーを指定する場合（Compound Sort Key）とは特性が異なり、各列を同等に扱う

- Interleaved Sort Keyが有効なケース
  - どのキーがWHERE句で指定されるか絞り切れないケース
  - 複数キーのAND条件で検索されるケース

Interleaved Sort Keyのデータ配置イメージ

Compound Sort Key

DeptId	LocId
1	A
1	B
1	C
1	D
2	A
2	B
2	C
2	D

DeptId	LocId
3	A
3	B
3	C
3	D
4	A
4	B
4	C
4	D

DeptId = 1 -> 1 block
LocId = C -> 4 block

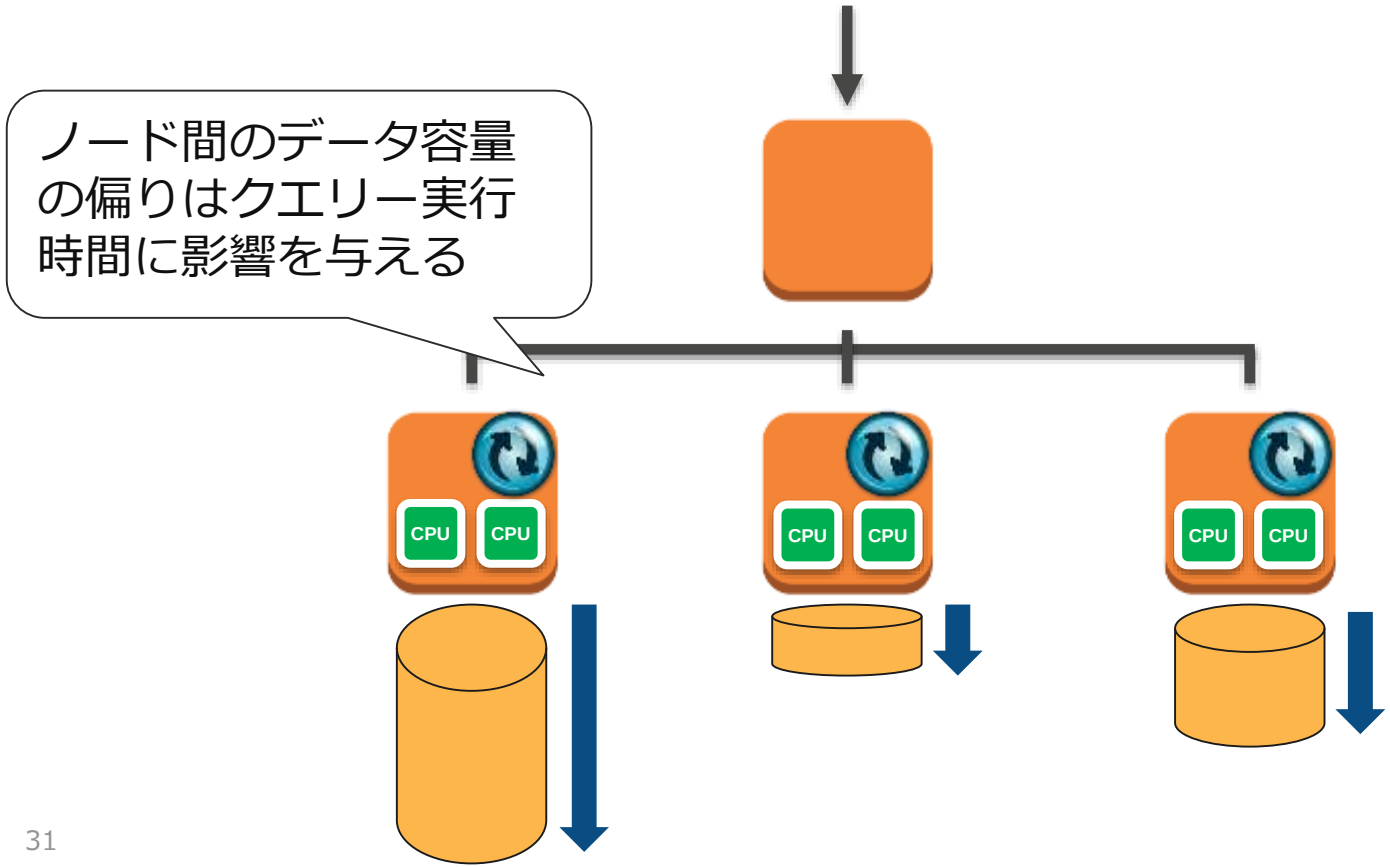
Interleaved Sort Key

DeptId	LocId
1	A
1	B
2	A
2	B
1	C
1	D
2	C
2	D

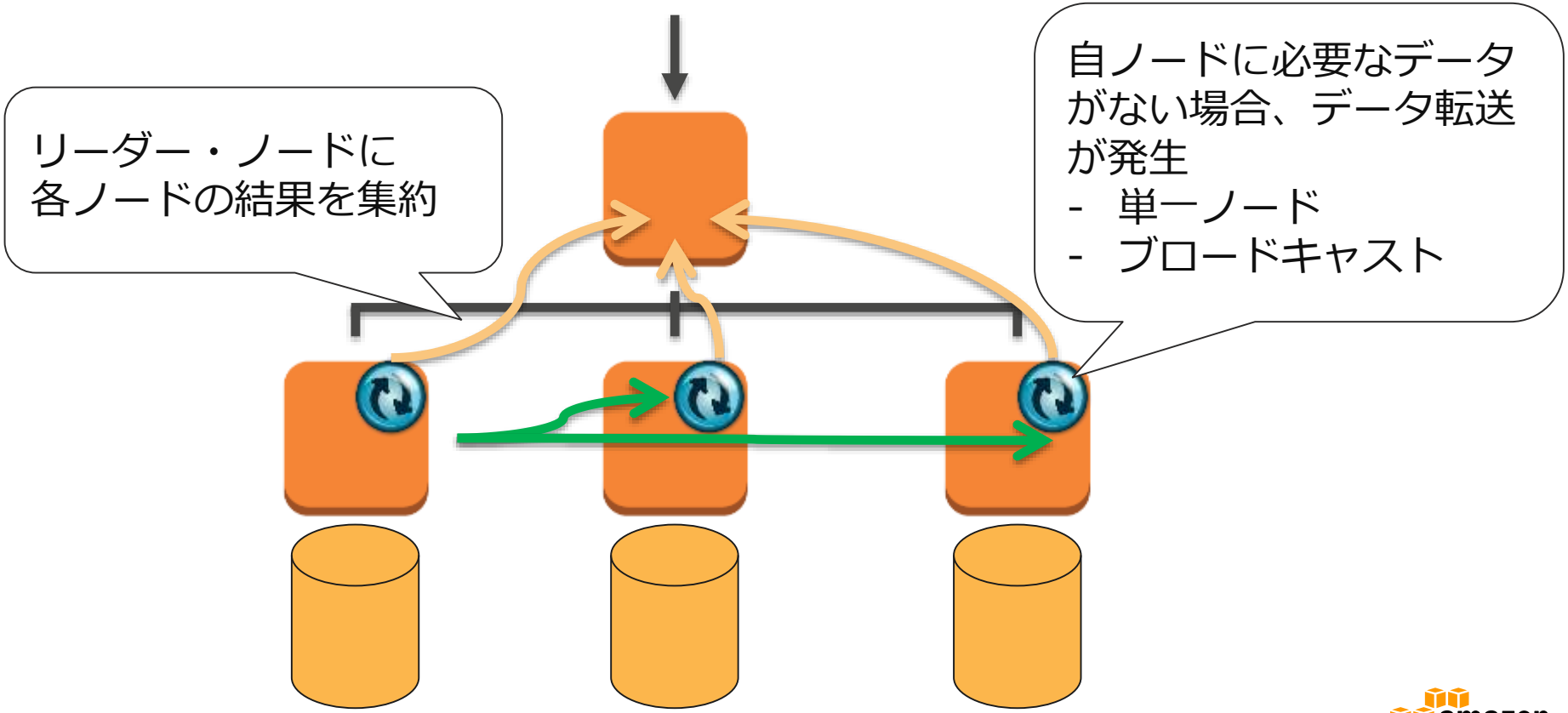
DeptId	LocId
3	A
3	B
4	A
4	B
3	C
3	D
4	C
4	D

DeptId = 1 -> 2 block
LocId = C -> 2 block
DeptId=1 and LocId=C-> 1 block

データの平準化:各ノードのデータサイズが著しく異なるとパフォーマンスに影響が出る



データの転送を最小限にする



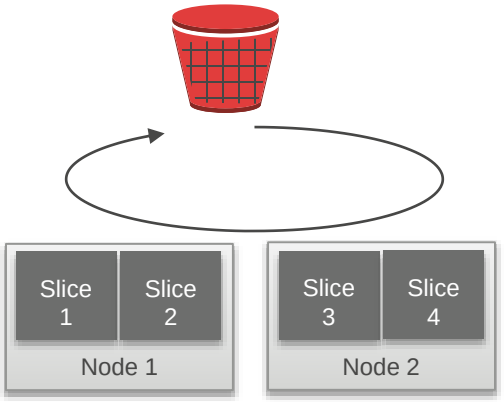
ディストリビューションの選択

CREATE TABLE t(…)

DISTSTYLE { EVEN | KEY | ALL }

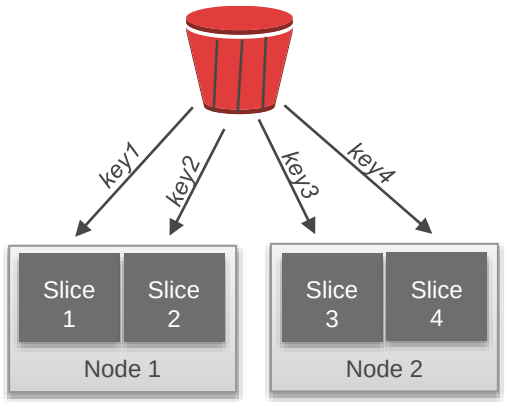
EVEN

ラウンドロビンで均一分散
（※デフォルト）



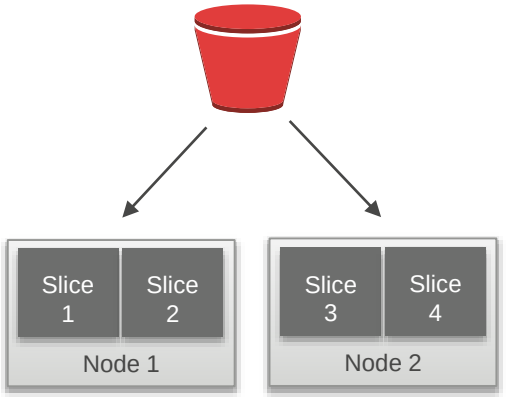
KEY(DISTKEY)

同じキーを同じ場所に



ALL

全ノードにデータをコピー



amazon web services

EVEN vs. DISTKEY（1）

- EVEN

```
select trim(name) tablename, slice,
sum(rows)
from stv_tbl_perm where name='part'
group by name, slice
order
 tablename | slice |   sum
-----------+-------+---------
 part      |     0 | 1600000
 part      |     1 | 1600000
…
 part      |   126 | 1600000
 part      |   127 | 1600000
```

各スライスに均等に分散

- DISTKEY=p_partkey

```
 tablename | slice |   sum
-----------+-------+---------
 part      |     0 | 1596925
 part      |     1 | 1597634
…
 part      |   126 | 1610452
 part      |   127 | 1596154
```

キーのカーディナリティに依存

EVEN vs. DISTKEY（2）

- DISTKEY = p_brand

```
 tablename | slice |    sum
-----------+-------+---------
 part      |     0 |       0
 part      |     1 |       0
 part      |     2 |       0
 part      |     3 |       0
 part      |     4 | 8193350
…
 part      |   118 | 8193342
 part      |   119 |       0
 part      |   120 | 16384823
 part      |   121 | 8191943
```

カーディナリティの低い
カラムでは、データの極端な
偏りが生じる場合がある
= クエリー処理効率の低下

ALL

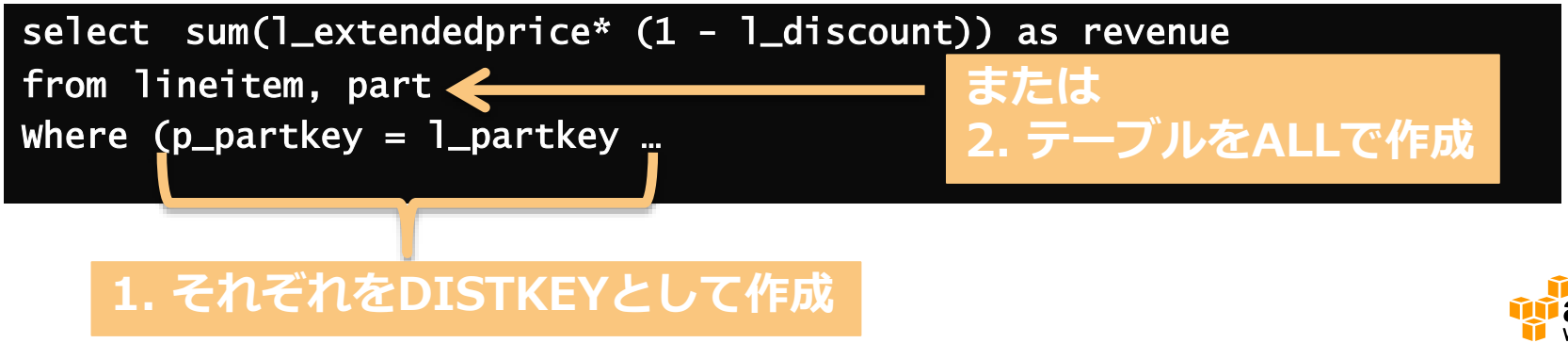
- 全レコードが各ノードの特定スライスに集約

```
 tablename | slice |    sum
-----------+-------+---------
 part      |     0 |204800000
 part      |     1 |         0
 part      |     2 |         0
 part      |     3 |         0
 part      |     4 |         0
…
 part      |    96 |204800000
 part      |    97 |         0
 part      |    98 |         0
…
```

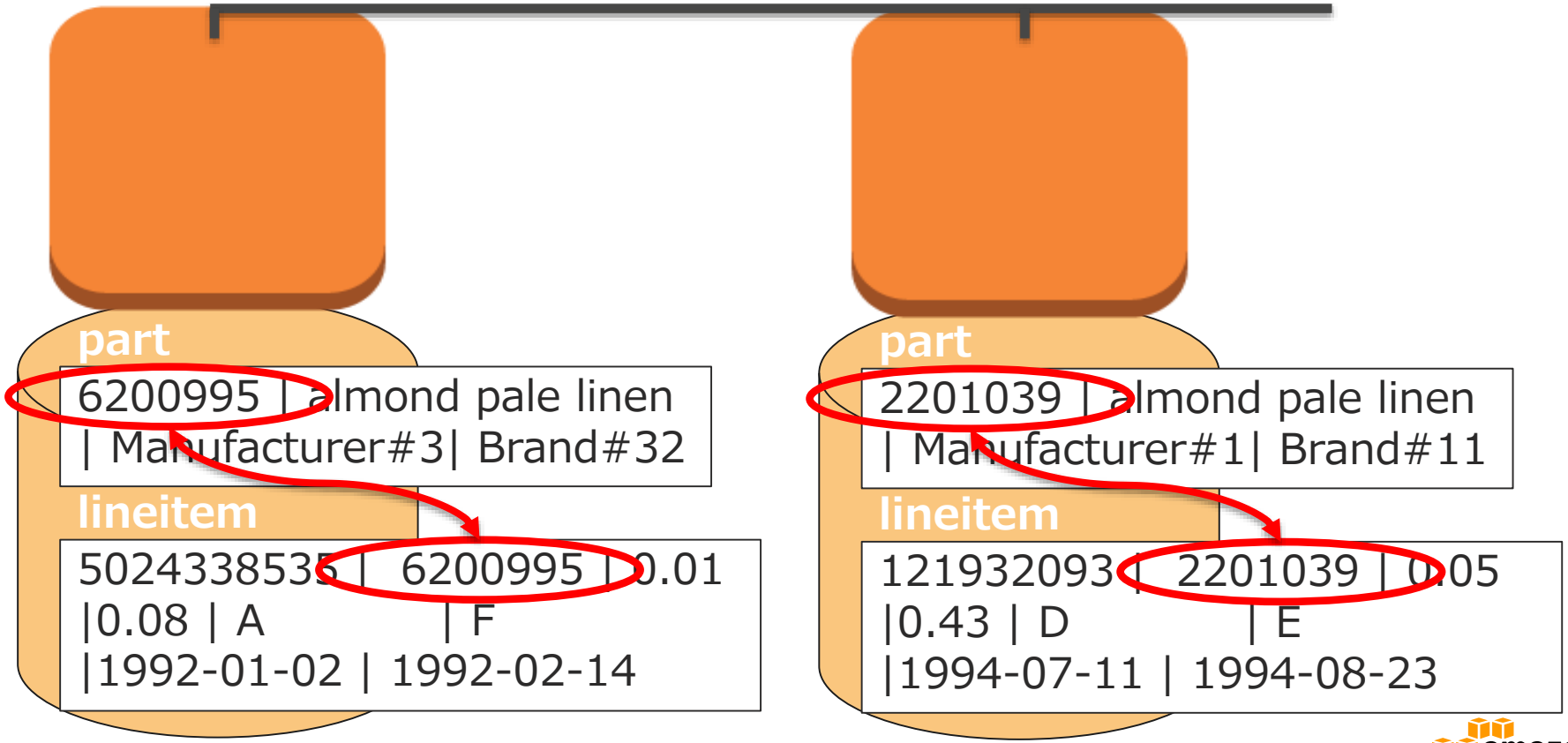
各ノードの先頭スライスに
全レコードが格納される。

コロケーション（1）

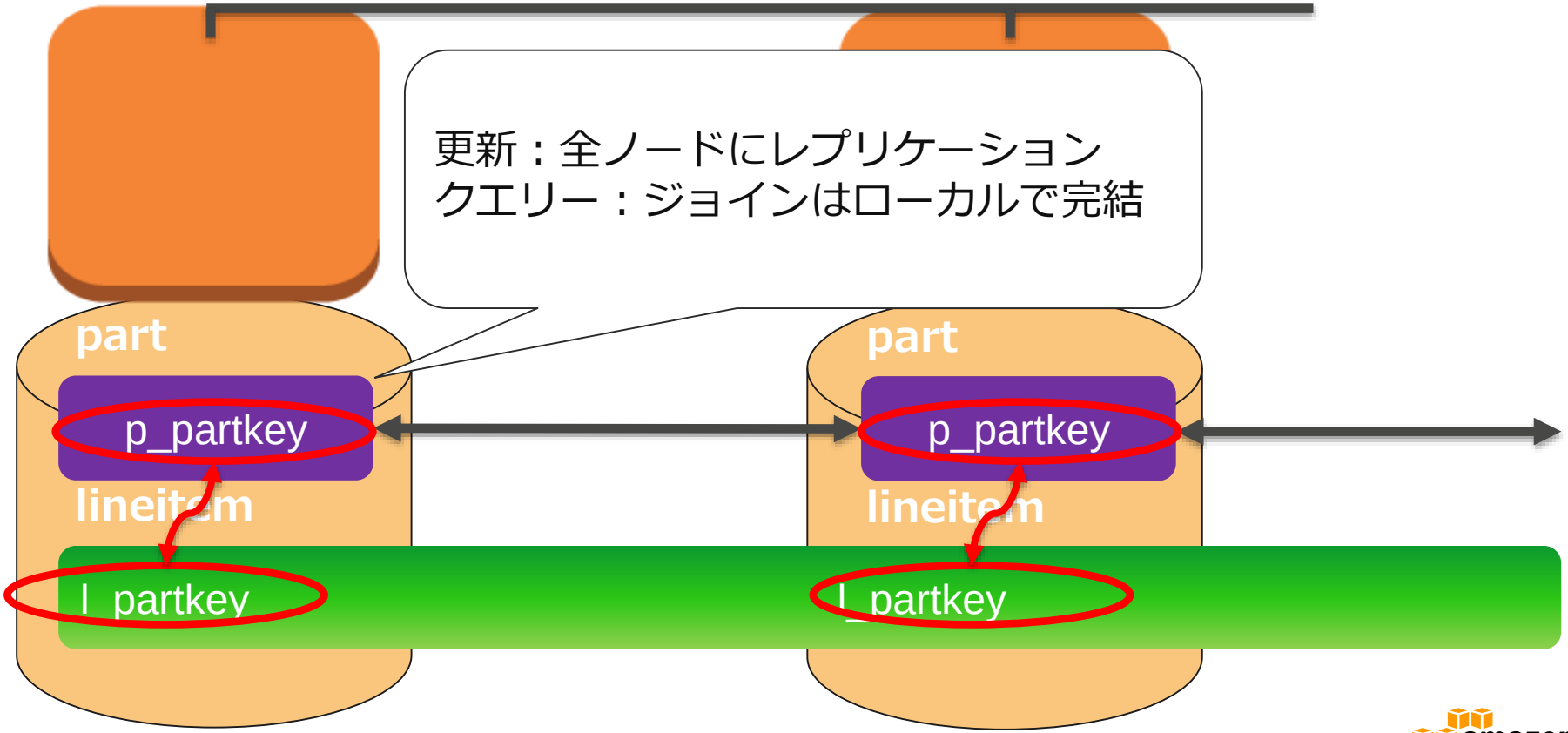
- 関連するレコードのコロケーション
  - ジョイン対象となるレコードを同一ノードに集める
- コロケーションの方法
  1. ジョインに使用するカラムをDISTKEYとして作成　**または**
  2. 分散方式 ALLでテーブルを作成（マスター・テーブルなど）



コロケーション（2）：DISTKEY



コロケーション（3）：ALL



テーブル設計のポイント

Tips

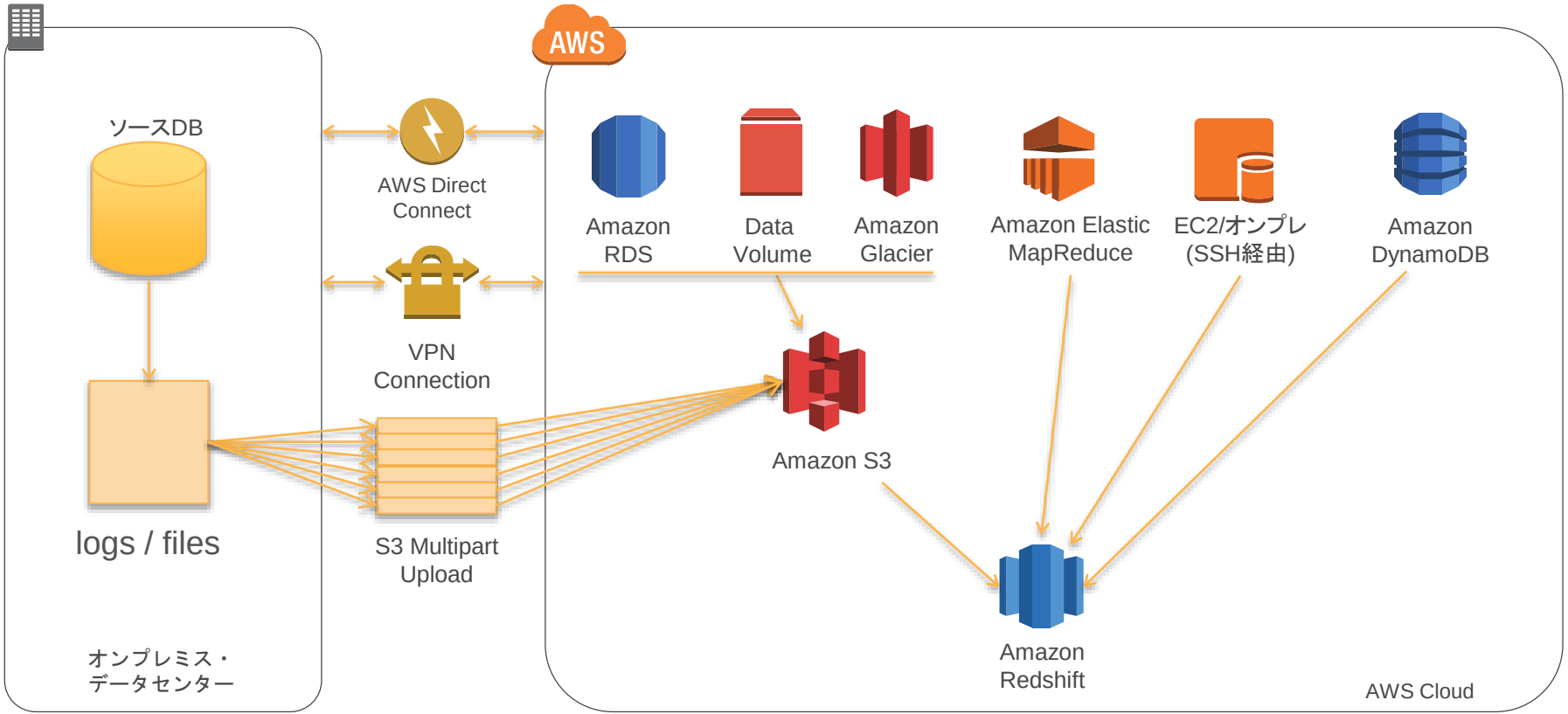
- ディスクIOを最小にする
  - 適切な型の選択
  - 適切な圧縮アルゴリズム
  - ソートキーの設定

- ネットワーク転送を最小にする
  - 小規模なテーブル（マスター・テーブル）はALLで作成する
  - 多くのテーブルはEVENで作成するだけで十分なパフォーマンスが出ることが多い
  - ジョインのパフォーマンスを最適化するにはジョイン対象のキーをDISTKEYで作成（コロケーション）
  - 大福帳のようなジョイン済（非正規化）表はEVENで分散

Agenda

- Amazon Redshiftとは？
- パフォーマンスを意識した表設計
- **Amazon Redshiftの運用**
- Workload Management (WLM)
- パフォーマンスチューニングテクニック Top 10
- まとめ

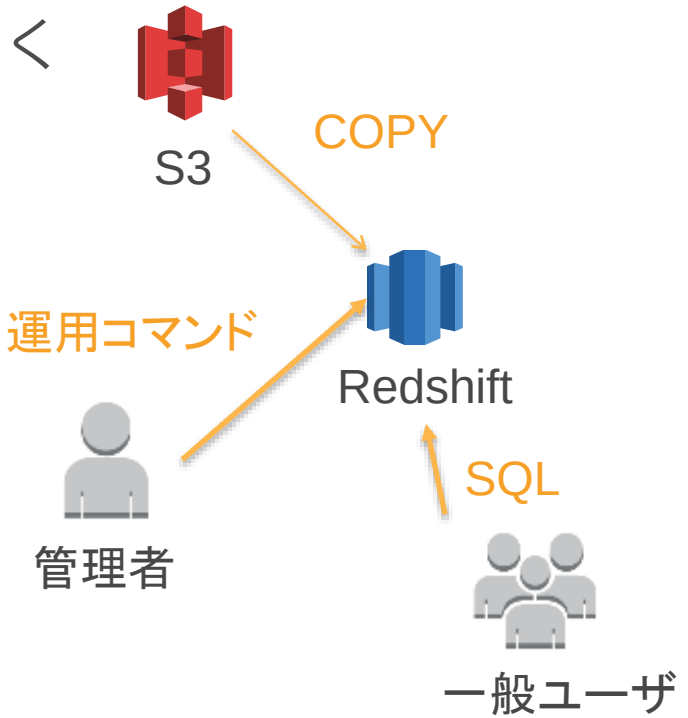
Amazon Redshiftへのデータ投入：オーバービュー



amazon web services

S3を起点としたRedshift運用の基本的な流れ

1. ロードするデータ（ファイル）をS3に置く
2. COPYコマンドでデータを高速ロード
3. Analyze＆Vacuumを実行
4. バックアップ（SNAPSHOT）を実行
5. SQLを投入して利用開始（1.へ戻る）



S3からデータをCOPYする

- ファイルをS3のバケットに置く
  - カンマや｜等で区切られたテキストファイル形式(delimiterオプションで指定)
  - **文字コードはUTF-8(デフォルト）とUTF-16をサポート**
  - ファイルサイズが大きい場合は圧縮し（後述）、マルチパートアップロードする
- Redshiftに接続してcopyコマンドを実行
  - S3にアクセスするためのアクセスキーが必要（:IAMロールでの対応可能に ）
  - 別リージョン内のS3バケットからのCOPYも可能 (REGIONオプションを指定)
- 自動圧縮される
  - 列にエンコーディング定義がなく、かつ1行も導入されていない場合に実施される
  - COMPUPDATE OFFオプションを指定すると自動圧縮無しでCOPY

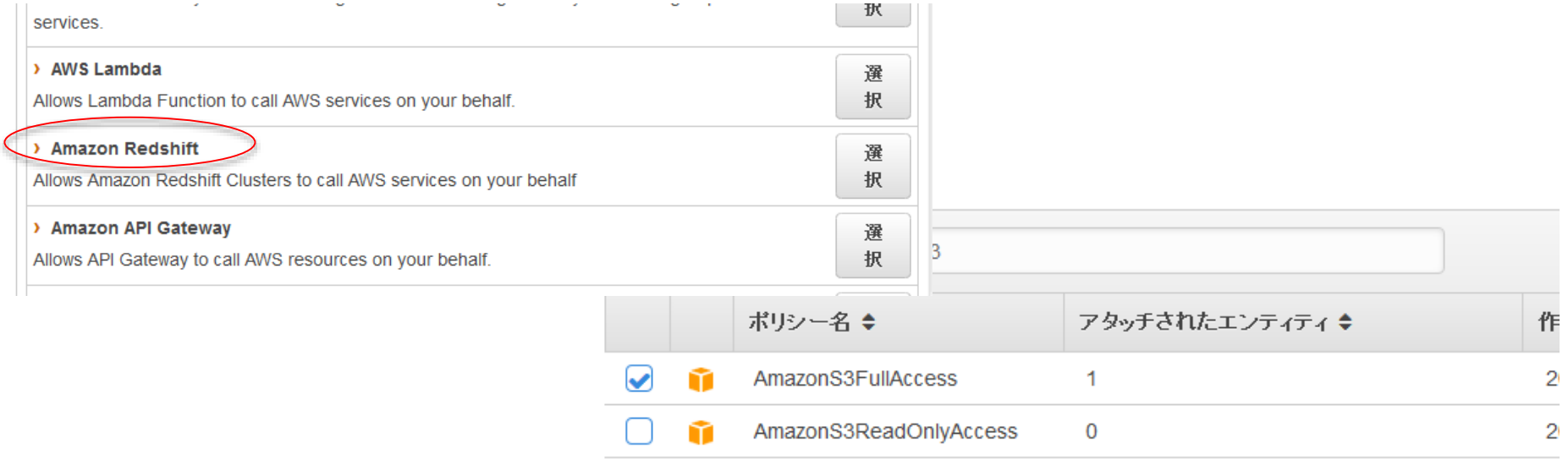
New!!

```
copy customer from 's3://mybucket/customer/customer.tbl'
  credentials 'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>'
  delimiter '|'
```

IAMロールに対応：COPY/UNLOAD

New!!

- Redshiftクラスターに対してIAMロールを付与して、その付与したロールでCOPY、UNLOADができるようになりました



COPYの速度を上げるには？

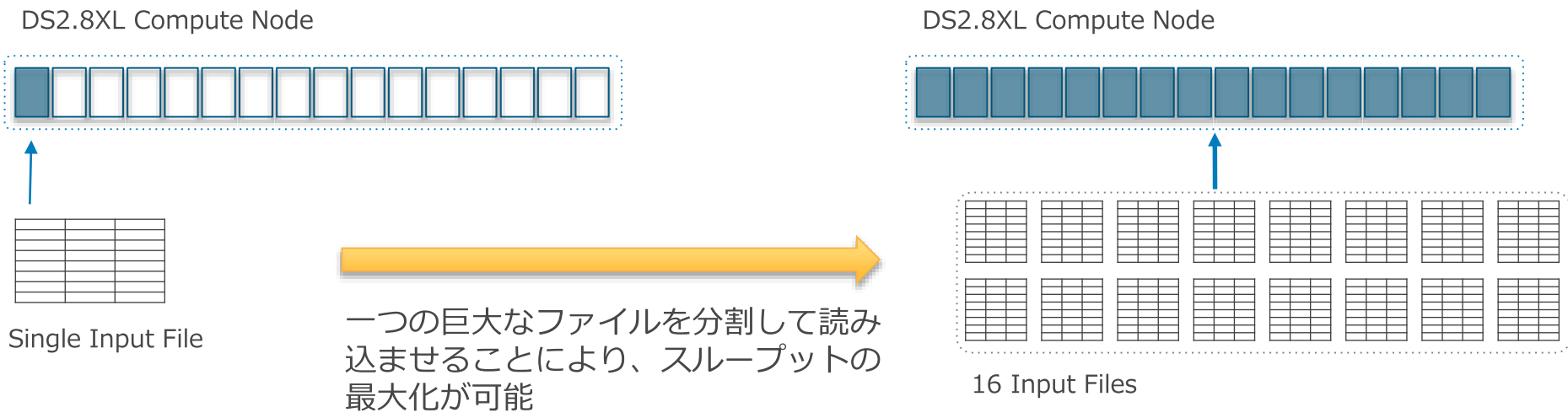
New!!

- 元ファイルを圧縮する(gzip,bzip2もしくはlzo)
  - COPYでgzip , bzip2 もしくはlzoオプションを指定
- ファイルを分割する（スライス数の倍数が最適）
  - 並列にロードされるため高速にロード可能
- ファイル名は”customer.tbl.1.gz”,”customer.tbl.2.gz”のように、指定した名前で前方一致出来るように作成

```
copy customer from 's3://mybucket/customer/customer.tbl'
  credentials 'aws_access_key_id=<access-key-id>;aws_secret_access_key=<secret-access-key>'
  gzip
  delimiter '|'
```

データロード時のスループット最大化

ファイルを分割する（スライス数の倍数が最適）
並列にロードされるため高速にロード可能

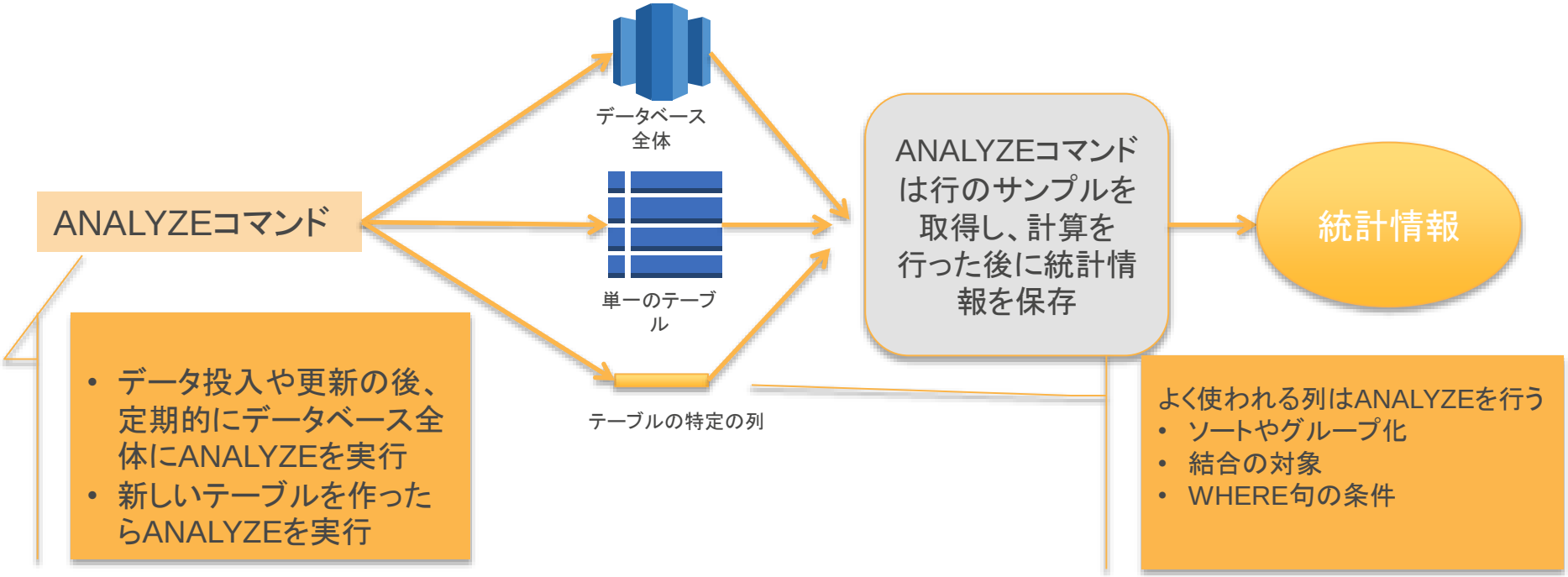


制約について

- Redshiftには制約が存在しない
  - ユニーク制約、プライマリーキー、外部キー、検査制約が無い
  - ユーザ側の工夫でユニーク性を担保する
    - 例）一旦データをテンポラリ表にインサートもしくはCOPYし、SELECT DISTINCTしたデータをインサートする
  - **制約やプライマリーキーの作成は可能。**作成する事でオプティマイザーにデータの特性情報を伝えることが可能

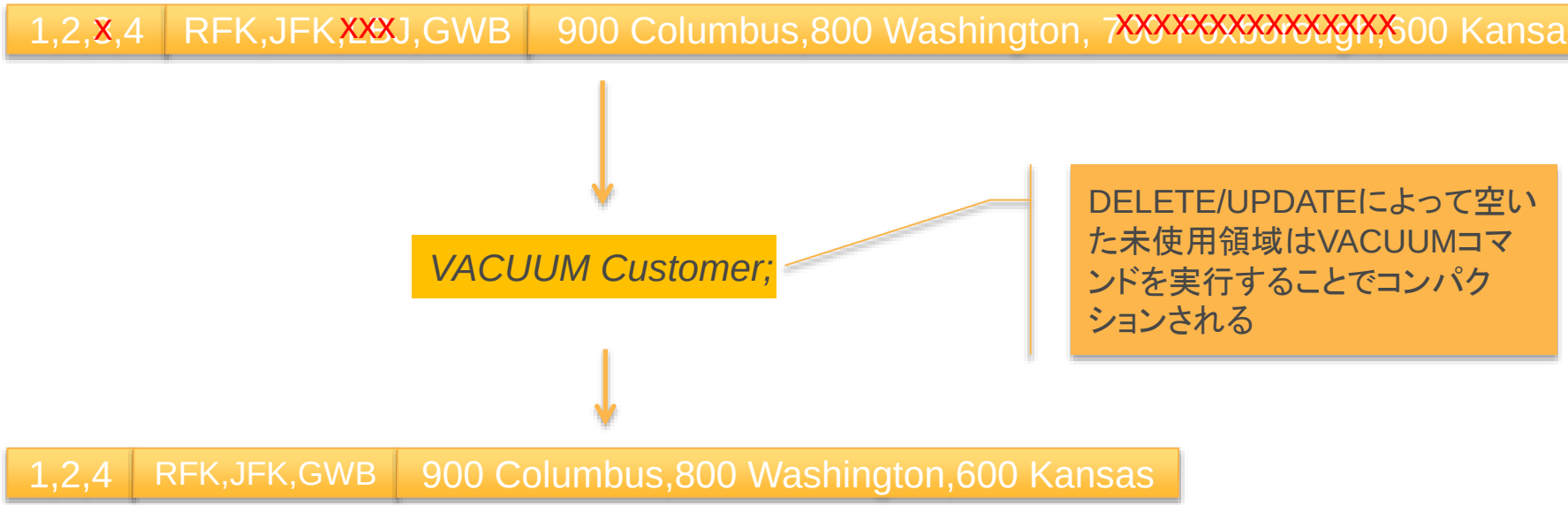
テーブルのANALYZE

- 統計情報はクエリプラン決定の元データとして利用される
- ANALYZEコマンドで統計情報を最新に保つことで最適なパフォーマンスを維持



テーブルのVACUUM

- Redshiftのデータ更新は“追記型”
- 削除しても削除がマークされるだけでディスク上にはデータが残っている
- VACUUMコマンドで不要領域を削除（コンパクション）し、同時にソート順にデータを並べ替える



amazon web services

VACUUMコマンド

```
VACUUM [ FULL | SORT ONLY | DELETE ONLY | REINDEX ] [ [ table_name ]
[ TO threshold PERCENT ] ]
```

- 通常はFULLを実行（コンパクション＆ソート）
  - ✓ コンパクションだけ実行するにはDELETE ONLY
  - ✓ ソートだけ実行するにはSORT ONLY
- Interleaved Sortした表にはREINDEXを指定
  - ✓ コンパクション＆Interleaved Sort順に並べ替えを実行
- TO threshold PERCENT 

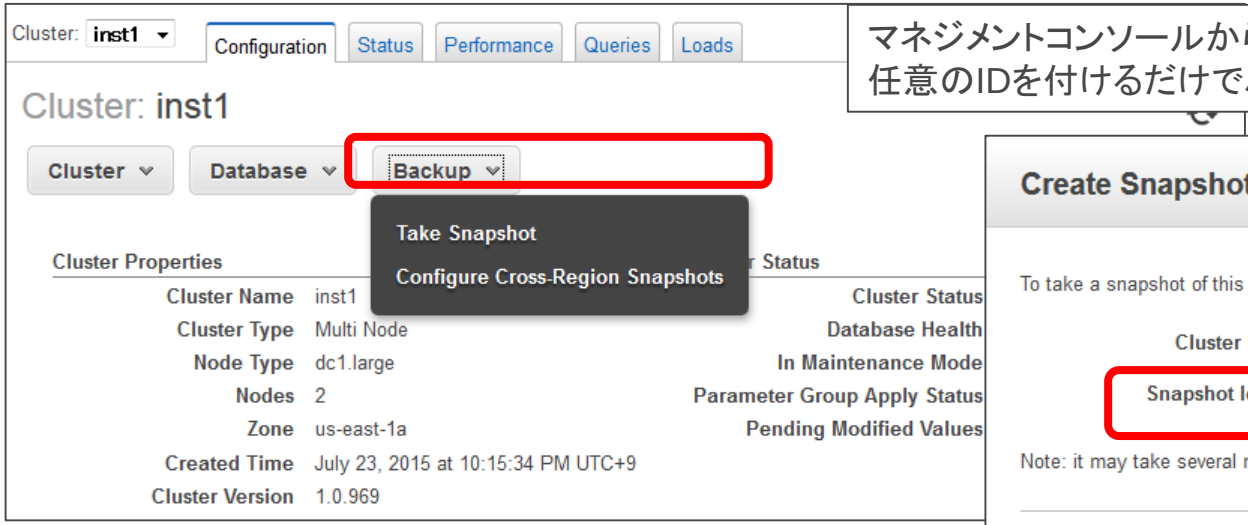
  - ✓ ソートする割合をパーセントで指定可能に

ALTER TABLE APPEND

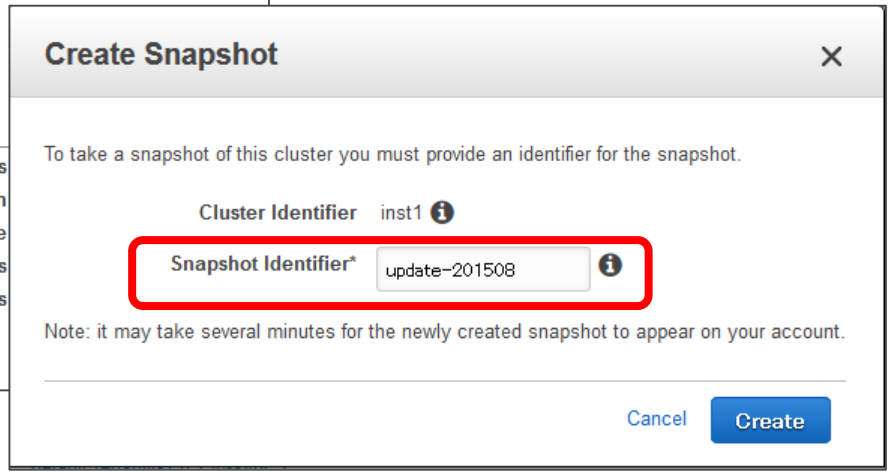
- ソース表のデータ全体をターゲット表にデータを「移動（MOVE）」する
- データを複製するのではなく移動するため、同様の CREATE TABLE AS または INSERT の INTO オペレーションよりもはるかに**高速**
- オペレーションが完了するとすぐに自動的にコミットします。ロールバックすることはできません。
- トランザクションブロック (BEGIN ... END) 内で ALTER TABLE APPEND を実行することはできません。

バックアップ機能 – スナップショット

- ディスクイメージをS3へバックアップ
  - 自動スナップショット
  - 手動スナップショット：ユーザが任意のタイミングで実行
  - テーブル単位でのリストア可能に New!!

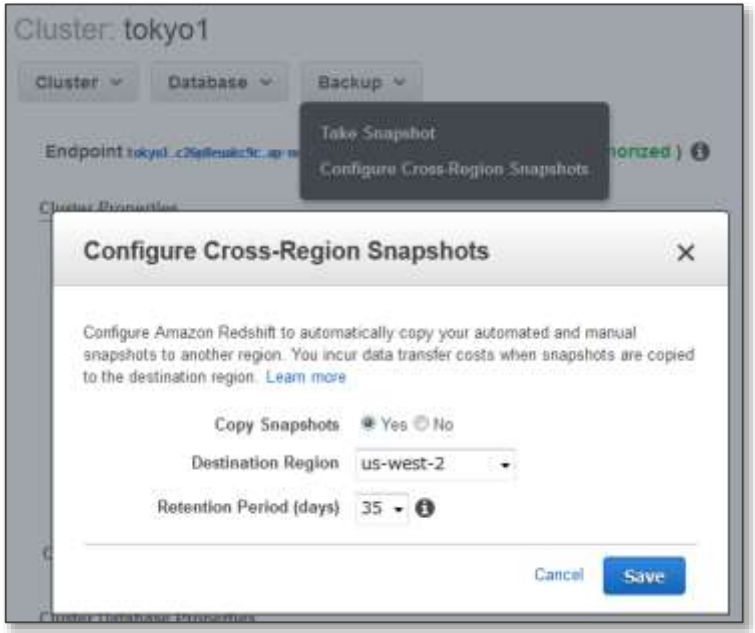


マネジメントコンソールから“Take Snapshot”を選択し、
任意のIDを付けるだけでバックアップ開始



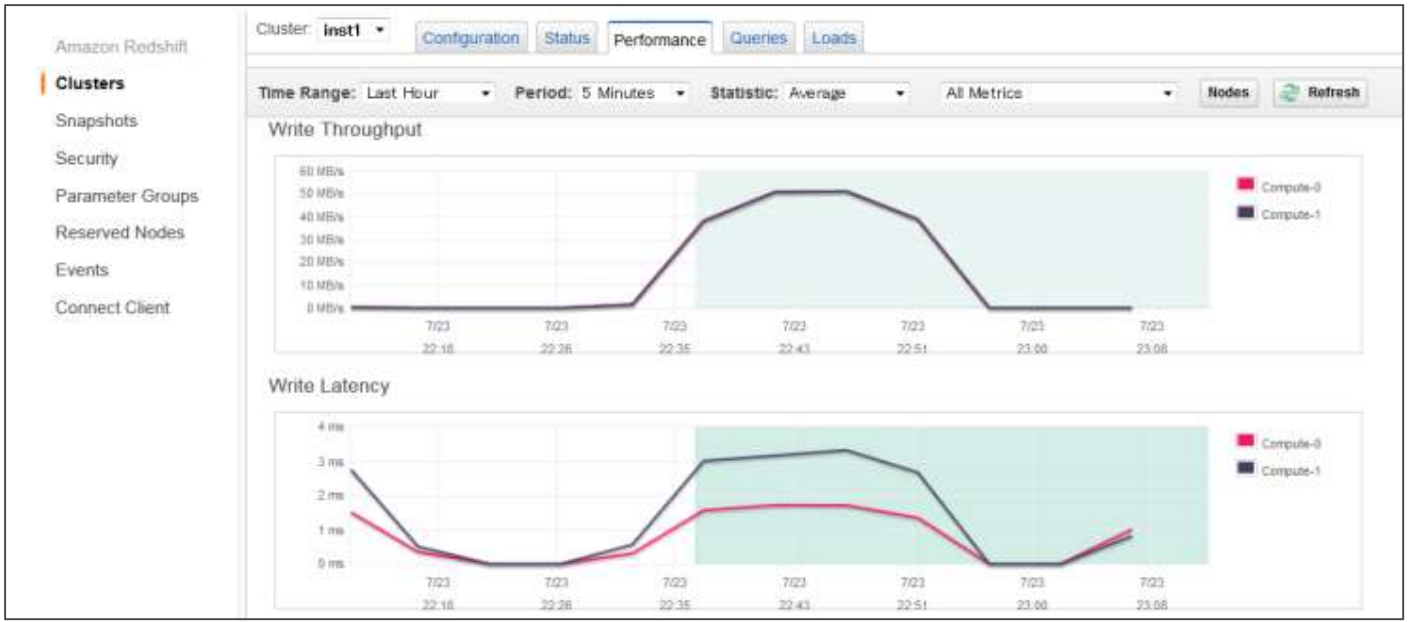
他リージョンへのスナップショット

- 既存クラスタのスナップショットを別リージョンに作成可能
- リテンション・ピリオド（保存期間）の指定も可能（最大35日）
- KMS暗号化済のスナップショット転送にも対応
- リージョン間のデータ転送費用が発生



Redshiftのモニタリング

- コンソールビルトインのGUI
  - リソース使用率、EXPLAIN、実行クエリー履歴 等
- API経由でデータ取得可能（CloudWatch）



実行クエリの確認

Actual タブで実行結果を各ステップ毎に確認可能

amazon web services

Agenda

- Amazon Redshiftとは？
- パフォーマンスを意識した表設計
- Amazon Redshiftの運用
- **Workload Management (WLM)**
- パフォーマンスチューニングテクニック Top 10
- まとめ

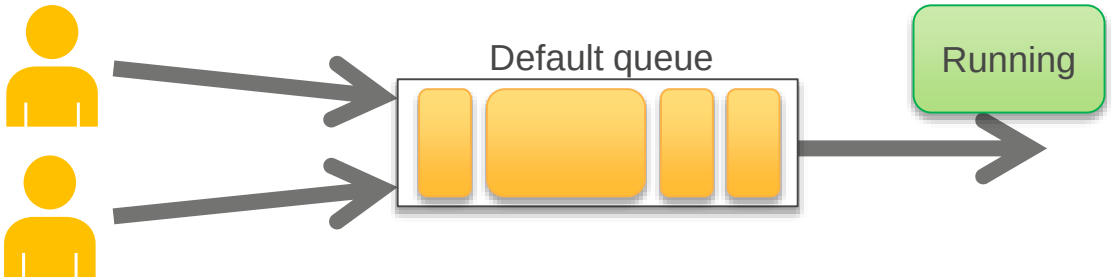
Workload Management (WLM)

- 実行に長い時間を用するクエリー（ロングクエリー）は、クラスタ全体のボトルネックとなり、ショートクエリーを待たせる可能性がある

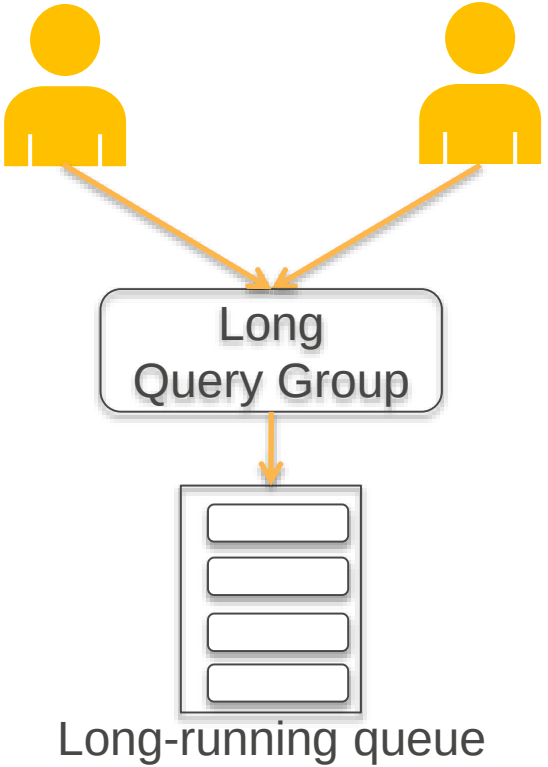
- WLMで用途ごとに、クエリー並列度の上限を設けた複数のキューを定義することでクエリー処理の制御が可能

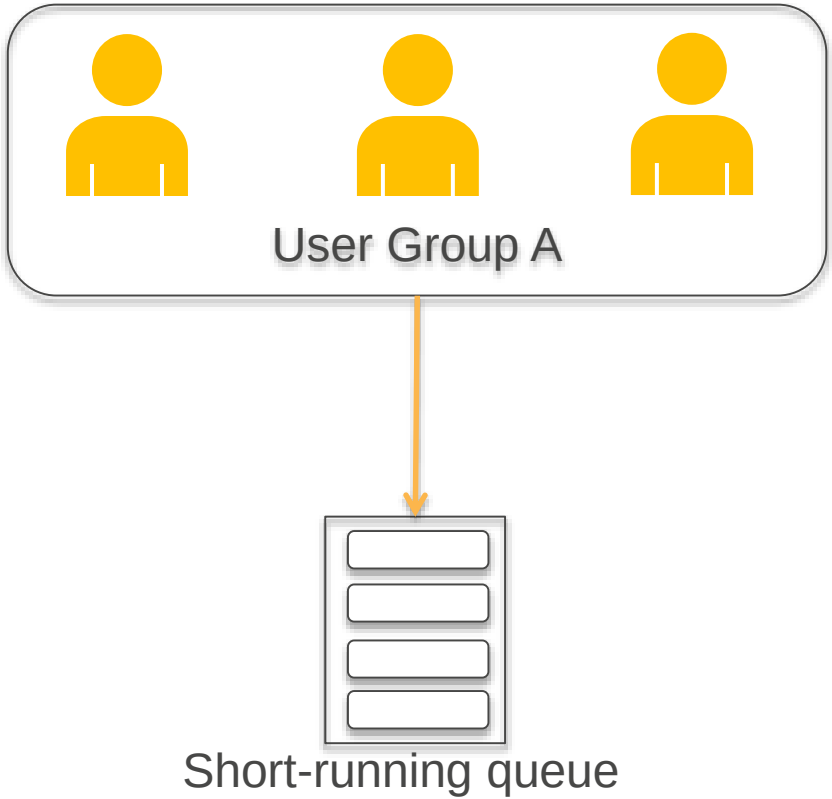
Workload Management

- 実行に長い時間を用するクエリー（ロングクエリー）は、クラスタ全体のボトルネックとなり、ショートクエリーを待たせる可能性がある
- WLMで用途ごとに、クエリー並列度の上限を設けた複数のキューを定義することでクエリー処理の制御が可能
- デフォルトでは、Redshiftクラスタは単一のキューで構成されている。



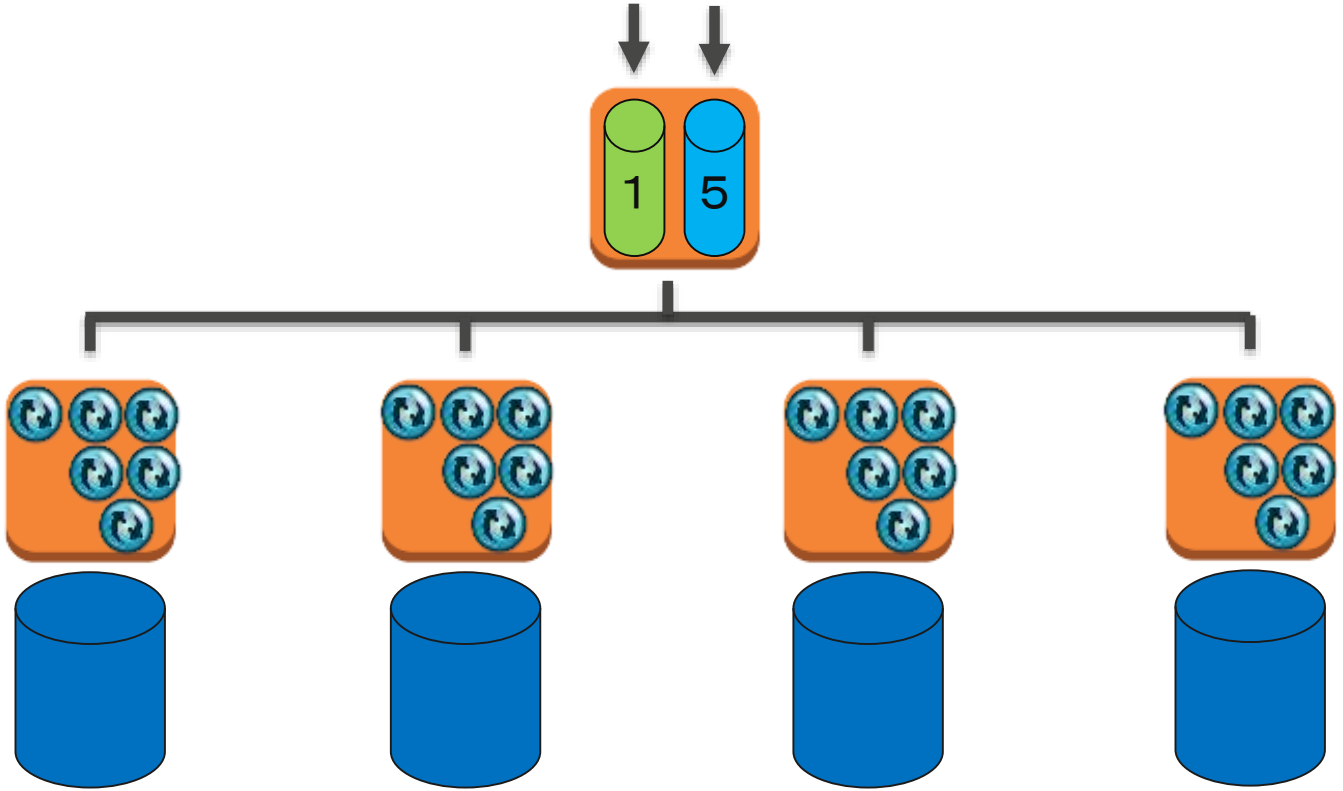
WLMの実装（１）





amazon web services

WLMの実装（２）



amazon web services

WLMの効果

- キュー単位でクエリー並列度を保障
  - メモリのアロケーションも指定可能
- 特定ユーザ（群）によるクラスタ占有を回避
  - 最大クエリー実行時間による制御も可能
- 並列度の増加は、必ずしも性能の向上には
  つながらない -> リソース競合の可能性

WLMパラメータとパラメータの動的変更

- 新しくWLMのパラメータにdynamicとstaticの区別が用意され、dynamicはRedshiftを再起動せずにパラメータ変更が可能に

- dynamic parameter
  - Concurrency(並列実行数),
  - Percent of memory to use (メモリ使用量)
- static parameter
  - User groups
  - User group wildcard
  - Query groups
  - Query group wildcard
  - Timeout

WLMの新機能
Queue hopping for time-out queries

New!!

- タイムアウトが発生した時にクエリーを停止するだけでなく、自動的に条件にマッチするキューにクエリーを投入（クライアントにはエラーは返却されない）

- 時間がかかり過ぎるクエリーをインタラクティブなクエリーから切り離し、別キューでゆっくり実行させる事が可能

Agenda

- Amazon Redshiftとは？
- パフォーマンスを意識した表設計
- Amazon Redshiftの運用
- Workload Management (WLM)
- **パフォーマンスチューニングテクニック Top 10**
- まとめ

パフォーマンスチューニングテクニック Top 10

http://aws.typepad.com/sajp/2015/12/top-10-performance-tuning-techniques-for-amazon-redshift.html

- 課題#1 – 間違った列の圧縮方式
  →IOを減らすために、適切な圧縮を列に指定する（ANALYZE COMPRESSION）
- 課題#2 – 偏ったテーブルデータ
  →適切な分散キーの選択
- 課題#3 – ソートキーの恩恵を受けられないクエリ
  → where句内に関数を使わない：　悪い例> where **cast** (date) = ‘ 2016-7-18’
- 課題#4 – 統計情報が無い表、もしくはVACCUMが必要な表
  →定期的なANALYZE、VACCUMの実施
- 課題#5 – とても大きなVARCHAR列を含む表
  → VARCHARカラム作成時に不必要に大きな値を指定しない

パフォーマンスチューニングテクニック Top 10

- 課題#6 – キュースロットをウエイトしているクエリ
  →適切なWLMの設定
- 課題#7 – ディスクベースのクエリー
  →クエリーが使用するメモリー量の増加（WLM_QUERY_SLOT_COUNT）
- 課題#8 – コミットキューのウエイト
  →コミット回数を減らす
- 課題#9 – 非効率的なデータロード
  →インプットファイルを分割する、圧縮する。 Insertではなく、COPYを使う
- 課題#10 – テンポラリ表の非効率的な利用
  →C(T)TASではなく、適切なテンポラリーテーブルを作りInsertする。

Tip: オプティマイザのアラートを利用する　STL_ALERT_EVENT_LOGビュー

http://docs.aws.amazon.com/ja_jp/redshift/latest/dg/r_STL_ALERT_EVENT_LOG.html

Agenda

- Amazon Redshiftとは？
- パフォーマンスを意識した表設計
- Amazon Redshiftの運用
- Workload Management (WLM)
- パフォーマンスチューニングテクニック Top 10
- **まとめ**

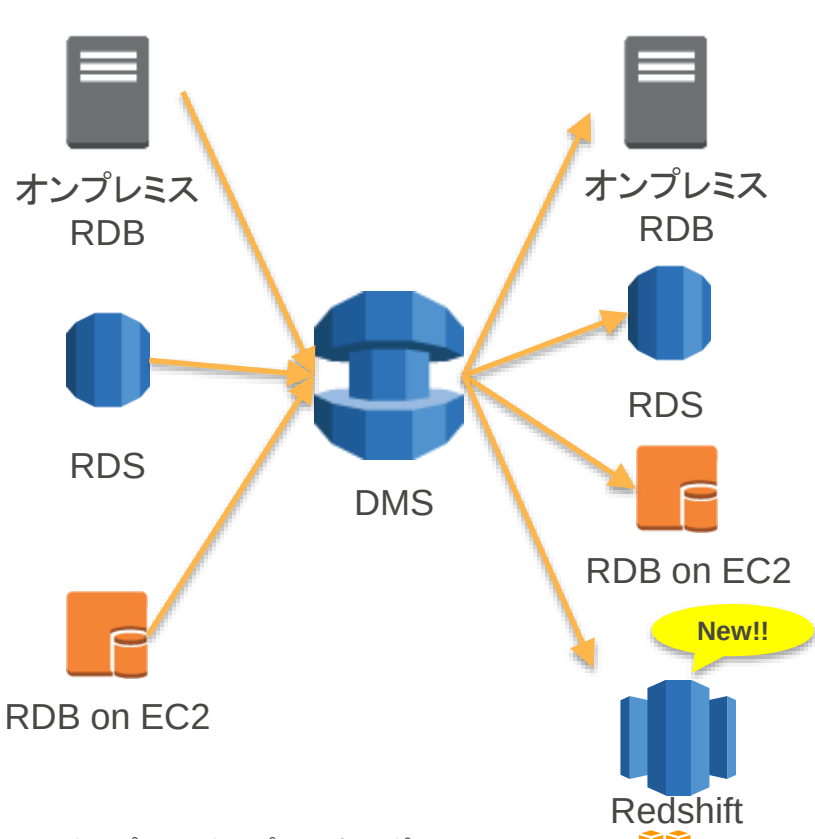
まとめ

Summary

- DWH的用途に特化したRDB
  - ペタバイト級まで拡張可能
- クラウドの良さを活かせるDWH
- マネージド・サービス
  - 機器セットアップやインストールの手間なし
  - バックアップ（スナップショット）が自動
  - その他運用に必要な各種機能（モニタリング、EXPLAIN等）をビルトインで提供
- チューニングポイント
  - ディスクIOの削減（圧縮、ソートキー）
  - ネットワーク通信の削減（分散の調整）
  - Workload Management
  - パフォーマンスチューニングテクニック Top 10のチェック

AWS Database Migration Service（DMS）

- ターゲットとして**Redshift**が対応可能に New!!
- RDB間のデータ移行を支援する**サービス**
- **異機種間**のデータ移行も対応



※オンプレ to オンプレは未サポート

Redshift 参考資料

- ドキュメント
  - https://aws.amazon.com/jp/documentation/redshift/
- フォーラム
  - https://forums.aws.amazon.com/forum.jspa?forumID=155&start=0
- 新機能アナウンスメント
  - https://forums.aws.amazon.com/thread.jspa?threadID=132076&tstart=25
- Amazon Redshift Utils on github
  - https://github.com/awslabs/amazon-redshift-utils

Q&A

Webinar資料の配置場所

- AWS クラウドサービス活用資料集
  - http://aws.amazon.com/jp/aws-jp-introduction/

- AWS Solutions Architect ブログ
  - 最新の情報、セミナー中のQ&A等が掲載されています
  - http://aws.typepad.com/sajp/

公式Twitter/Facebook
AWSの最新情報をお届けします

@awscloud_jp

もしくは
http://on.fb.me/1vR8yWm

最新技術情報、イベント情報、お役立ち情報、
お得なキャンペーン情報などを日々更新しています！

ご参加ありがとうございました


amazon
web services