

ORACLE®

シバタツ流！ パラレル・クエリーの徹底活用と チューニングの極意

日本オラクル株式会社
テクノロジー製品事業統括本部
基盤技術本部 応用技術グループ
プリンシパルエンジニア
柴田竜典



 #odddtky

日本オラクル、今年最大の技術トレーニング・イベント

**Oracle DBA &
Developer Day 2013**

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント(確約)するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

OracleとJavaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。文中の社名、商品名等は各社の商標または登録商標である場合があります。

Program Agenda

- なぜ今パラレル実行なのか
- パラレル実行の実行計画を読もう
- チューニング・ケーススタディ

シバタツって誰？

- 日本オラクル株式会社
テクノロジー製品事業統括本部 基盤技術本部 応用技術グループ
プリンシパルエンジニア 柴田竜典
- Oracle Technology Network にて全5回で
『シバタツ流！ DWHチューニングの極意』を連載
 - これを読めばDWHデザインを間違わない！
 - すべての回に似顔絵がついている！

シバタツ

検索



もう少しプロフィール

- Oracle Exadata リリース当初から、お客様のSQLやデータを使用したPoC (Proof of Concept) を実施
 - 本番稼働しているたくさんのシステムのパフォーマンス・チューニングを経験
- 2010年には米オラクルの開発部門の一員としてサンフランシスコのヘッド・クォーターで勤務
 - 米国のお客様のPoCを実施しつつ、そこから見えてきたOracle Database のパフォーマンス課題の解決に取り組む



なぜ今 パラレル実行なのか

パラレル実行とは

- 1個のSQLを複数プロセス(複数CPU)で実行すること
 - パラレル問合せ: SELECT文をパラレルで実行する
 - パラレルDML: INSERT ... SELECT 文などのDMLをパラレルで実行する
 - パラレルDDL: CREATE INDEX 文などのDDLをパラレルで実行する
- 多重実行: 複数のSQLを複数プロセスで動かすこと
 - パラレル実行 ≠ 多重実行

どういふときにパラレル実行すべきか

- パラレル実行すべきとき
 - 大量の行にアクセスする
(= 大量データを処理する = CPUリソースが必要)
 - 多重度が低い
- シリアル実行すべきとき
 - 少数の行にしかアクセスしない
 - 多重度が高い

なぜ今パラレル実行なのか

Oracle7でパラレル実行が可能に

- 10年前くらいからデータウェアハウスでのOracle利用が増えてきたが、IOネックだったのでCPU使用率は性能に影響しなかった
- ここ数年でIOネックを解消する方法ができ、CPUを1個しか使わないとCPUネックになってしまう状況が増えてきた
 - Exadata
 - フラッシュ・ストレージ



パラレル実行しないと性能が出ない

パラレル実行の 実行計画を読もう

実行計画に書いてあるこれはなに？

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S	QC (RAND)
* 3	HASH JOIN BUFFERED		Q1,02	PCWP	
4	PX RECEIVE		Q1,02	PCWP	
5	PX SEND HASH	:TQ10000	Q1,00	P->P	HASH
6	PX BLOCK ITERATOR		Q1,00	PCWC	
7	TABLE ACCESS FULL	CUSTOMERS	Q1,00	PCWP	
8	PX RECEIVE		Q1,02	PCWP	
9	PX SEND HASH	:TQ10001	Q1,01	P->P	HASH
10	PX BLOCK ITERATOR		Q1,01	PCWC	
* 11	TABLE ACCESS FULL	SALES	Q1,01	PCWP	

リアルタイムSQL監視に書いてあるこれはなに？

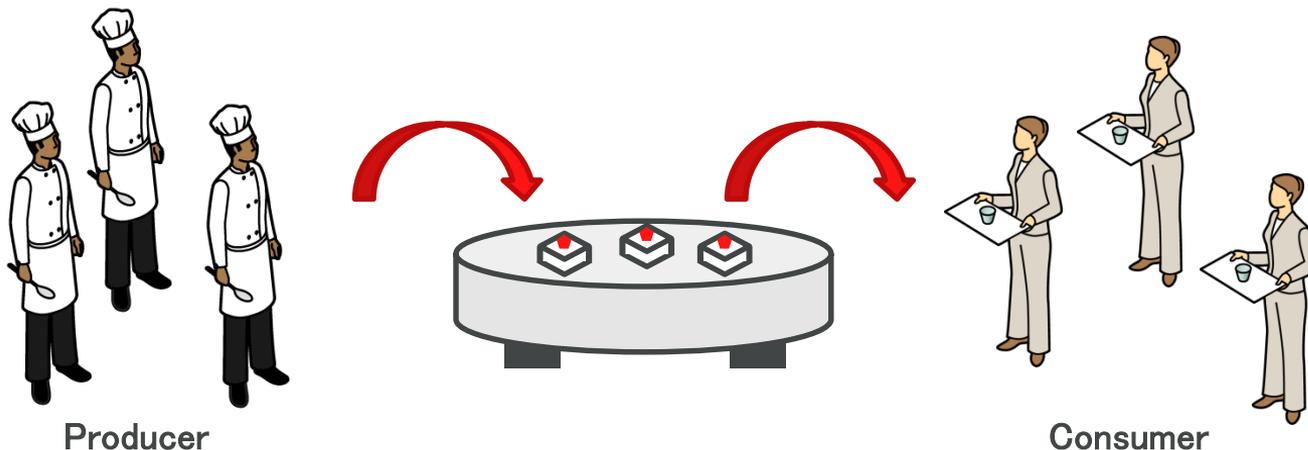
The screenshot shows the Oracle Enterprise Manager interface. On the left, the 'Active Reports' section displays '監視されたSQL実行の詳細' (Details of Monitored SQL Execution). The main area shows a plan table for plan hash value 2118738808. The plan table has columns for '操作' (Operation), '名前' (Name), '予測した...' (Estimated...), 'コスト' (Cost), and '時点' (Time). Red arrows point to the following operations in the plan table:

操作	名前	予測した...	コスト	時点
SELECT STATEMENT			0	
PX COORDINATOR			0	
PX SEND QC (RANDOM)	:TQ10002	405K	262	
HASH JOIN BUFFERED		405K	263	
PX RECEIVE		56K	113	
PX SEND HASH	:TQ10000	56K	113	
PX BLOCK ITERATOR		56K	113	
TABLE ACCESS FULL	CUSTOMERS	56K	113	
PX RECEIVE		405K	149	
PX SEND HASH	:TQ10001	405K	149	
PX BLOCK ITERATOR		405K	149	
TABLE ACCESS FULL	SALES	405K	149	

Producer-Consumerパターン

パラレル実行で使われているデザイン・パターン

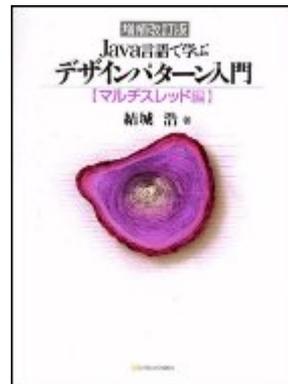
- Producer(生産者)が作ったケーキをテーブルに置き、Consumer(消費者)がテーブルのケーキを食べる



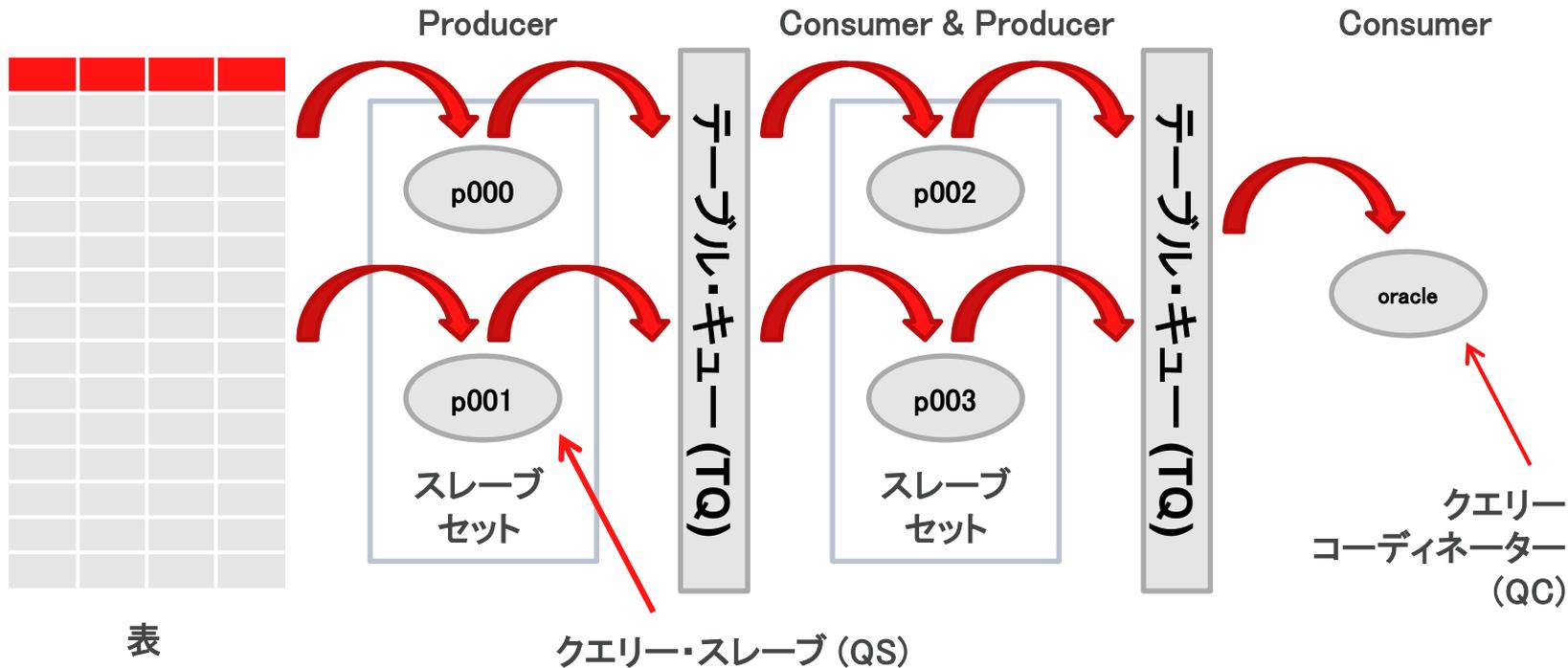
Producer-Consumerパターンのメリット

- Producerは生産することに、Consumerは消費することに専念できる
 - Producerが自分でケーキを食べると、食べている間に調理場が暇になる
- Producerの生産はConsumerの消費の遅れに影響しない
 - テーブルがない状況でConsumerの消費が遅れると、Producerはケーキを持って待っていないといけない

参考: 結城浩『増補改訂版 Java言語で学ぶデザインパターン入門
マルチスレッド編』(ソフトバンククリエイティブ)



パラレル実行のProducer-Consumerパターン



もう一度実行計画を見てみよう

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S	QC (RAND)
* 3	HASH JOIN BUFFERED		Q1,02	PCWP	
4	PX RECEIVE 		Q1,02	PCWP	
5	PX SEND HASH	:TQ10000	Q1,00	P->P	HASH
6	PX BLOCK ITERATOR		Q1,00	PCWC	
7	TABLE ACCESS FULL	CUSTOMERS	Q1,00	PCWP	
8	PX RECEIVE		Q1,02	PCWP	
9	PX SEND HASH 	:TQ10001	Q1,01	P->P	HASH
10	PX BLOCK ITERATOR		Q1,01	PCWC	
* 11	TABLE ACCESS FULL	SALES	Q1,01	PCWP	



もう一度リアルタイムSQL監視を見てみよう

The image displays the Oracle Enterprise Manager Active Reports interface. The main window shows the details of a monitored SQL statement (SQL ID: b22cdvnd77z) executed on 2013年10月28日. The execution plan is visible, showing a HASH JOIN BUFFERED operation. A red box highlights the 'Plan' tab in the navigation bar.

A zoomed-in view of the execution plan is shown in the foreground, titled '計画ハッシュ値 2118738808'. The plan consists of the following operations:

- SELECT STATEMENT
- PX COORDINATOR
- PX SEND QC (RANDOM) :TQ10002
- HASH JOIN BUFFERED
- PX RECEIVE
- PX SEND HASH :TQ10000
- PX BLOCK ITERATOR
- TABLE ACCESS FULL CUSTOMERS
- PX RECEIVE
- PX SEND HASH :TQ10001
- PX BLOCK ITERATOR
- TABLE ACCESS FULL SALES

Red arrows point to the 'PX SEND QC (RANDOM)', 'HASH JOIN BUFFERED', 'PX RECEIVE', and 'PX SEND HASH' operations in the zoomed view.

TQ列

- フォーマット: Q\${IDofQC}, \${IDofTQ} 例) Q01, 02
- 各ステップで処理された行が渡されるテーブル・キューのID
- テーブル・キューは抽象概念
 - 共有プールのメモリー上に読み取ったデータがコピーされ.....というようなSGAを経由して2回コピーするような実装ではない
- 実際には各プロセスがそれぞれキューを持っており、そこに置かれたデータをキュー・リファレンスという番号を使ってリンクしている
- 読み取ったデータは各プロセスのPGAからPGAに直接コピーされる

IN-OUT列

プロセス間通信しない方式

- PCWP:
Parallel Combined with Parent
 - 次のステップも同一のQSが行なう
- PCWC:
Parallel Combined with Child
 - 前のステップと同一のQSが行なう

プロセス間通信する方式

- P->P: Parallel to Parallel
 - QSが次のスレーブ・セットにデータを送る
- P->S: Parallel to Serial
 - QSがQCにデータを送る
- S->P: Serial to Parallel
 - QCがQSにデータを送る
 - このステップはシリアル実行

PQ Distrib 列

P->Pでの分散方式のみ抜粋

- HASH

- 結合キーをハッシュ分散させて各QSへ送る

- RANGE

- ソート・キーで分散させて各QSへ送る

- PART (KEY)

- パーティション・キーで分散させて各QSへ送る

- BROADCAST

- 表全体をすべてのQSに送る

結合時の分散方式

PQ Distrib 詳細

HASH / RANGE / PART (KEY) 分散方式

大きい表と大きい表を結合するとき

- 100人分の顧客名簿に載っている顧客からの売上げ明細（全体で1万件）を、10人がかりで抽出するときどうするか？
- 顧客名簿を「ア行で始まる人」「カ行で始まる人」……と10グループに分ける
- 売上げ明細を「顧客名がア行で始まる人」「顧客名がカ行で始まる人」……と10グループに分ける
- メンバーAはア行を担当する。メンバーBはカ行を担当する……

- 作業を分割できる（重複データがない）

BROADCAST分散方式

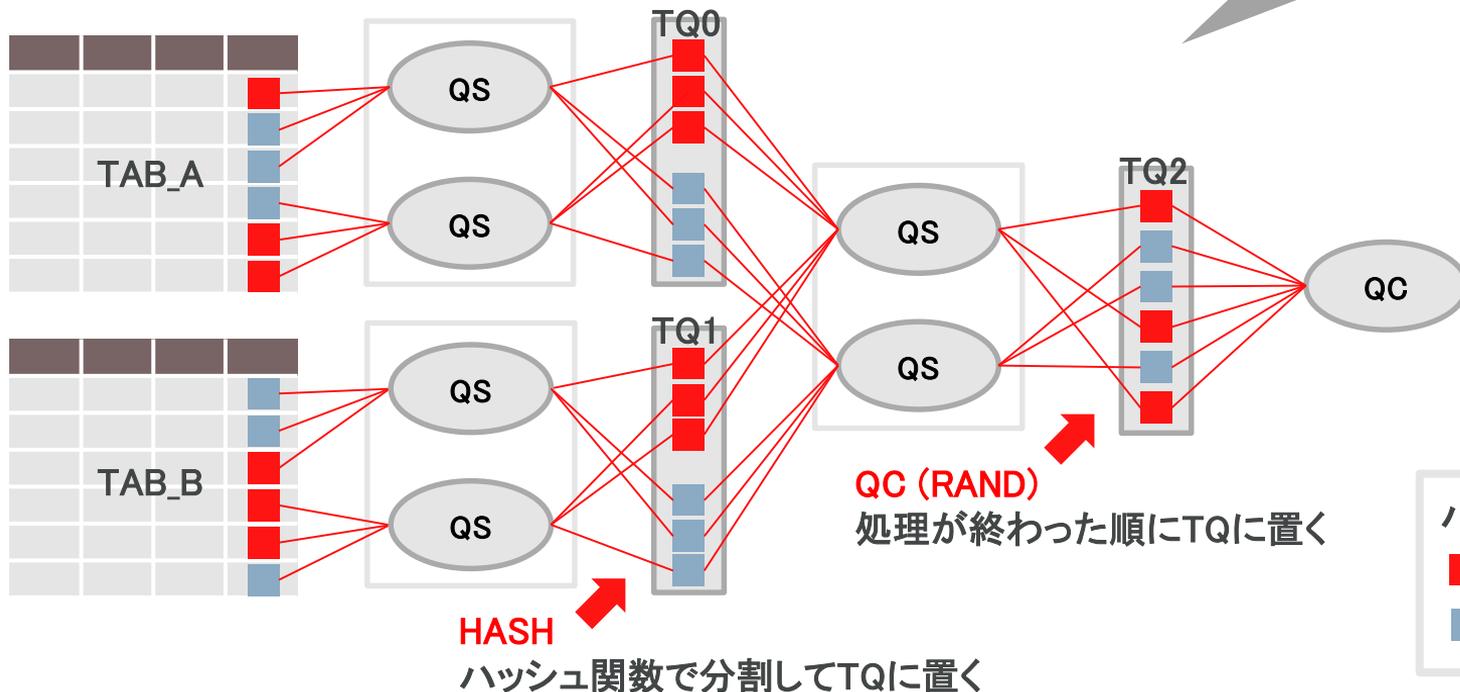
大きい表と小さい表を結合するとき

- 3人分の顧客名簿に載っている顧客からの売上げ明細（全体で1万件）を、10人がかりで抽出するときどうするか？
- 売上げ明細を無作為に10グループに分ける
- 全員が同じ顧客名簿を使って、メンバーAは自分の担当分を担当する。メンバーBは自分の担当分を担当する.....
- ハッシュ計算コストを省略できる

HASH分散方式

大きい表と大きい表を結合する場合

TQ, IN-OUT, PQ Distrib から
この図が書けるようになることが
パラレル実行の理解のカギ！



HASH分散方式の実行計画例

SQL Text

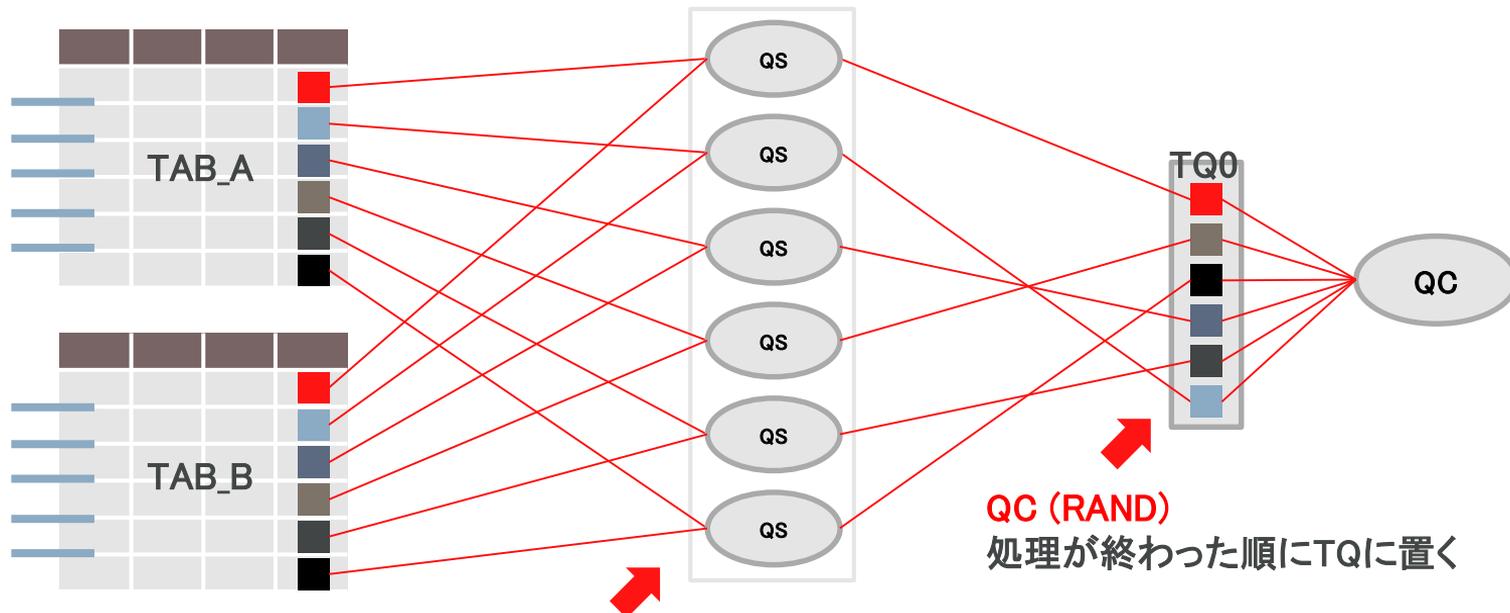
```
SELECT c.cust_email FROM sales s, customers c WHERE s.cust_id = c.cust_id AND s.amount_sold >= 1000
```

Execution Plan

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10002	Q1,02	P->S	QC (RAND)
* 3	HASH JOIN		Q1,02	PCWP	
4	<u>PX RECEIVE</u>		Q1,02	PCWP	
5	<u>PX SEND HASH</u>	:TQ10000	...	Q1,00	<u>HASH</u>
6	PX BLOCK ITERATOR		Q1,00	PCWC	
7	TABLE ACCESS FULL	CUSTOMERS	Q1,00	PCWP	
8	<u>PX RECEIVE</u>		Q1,02	PCWP	
9	<u>PX SEND HASH</u>	:TQ10001	Q1,01	P->P	<u>HASH</u>
10	PX BLOCK ITERATOR		Q1,01	PCWC	
* 11	TABLE ACCESS FULL	SALES	Q1,01	PCWP	

フル・パーティション・ウィズ結合

2個の表が両方とも結合キーでパーティションされている場合



パーティションされている = 分割済みなので、再分散しない

フル・パーティション・ウィズ結合時の実行計画例

SQL Text

```
SELECT s.prod_id, c.unit_price FROM sales s, costs c
WHERE s.prod_id = c.prod_id AND s.time_id = c.time_id
      AND s.promo_id = c.promo_id AND s.channel_id = c.channel_id
      AND s.amount_sold >= 1000 AND c.unit_price >= 1000
```

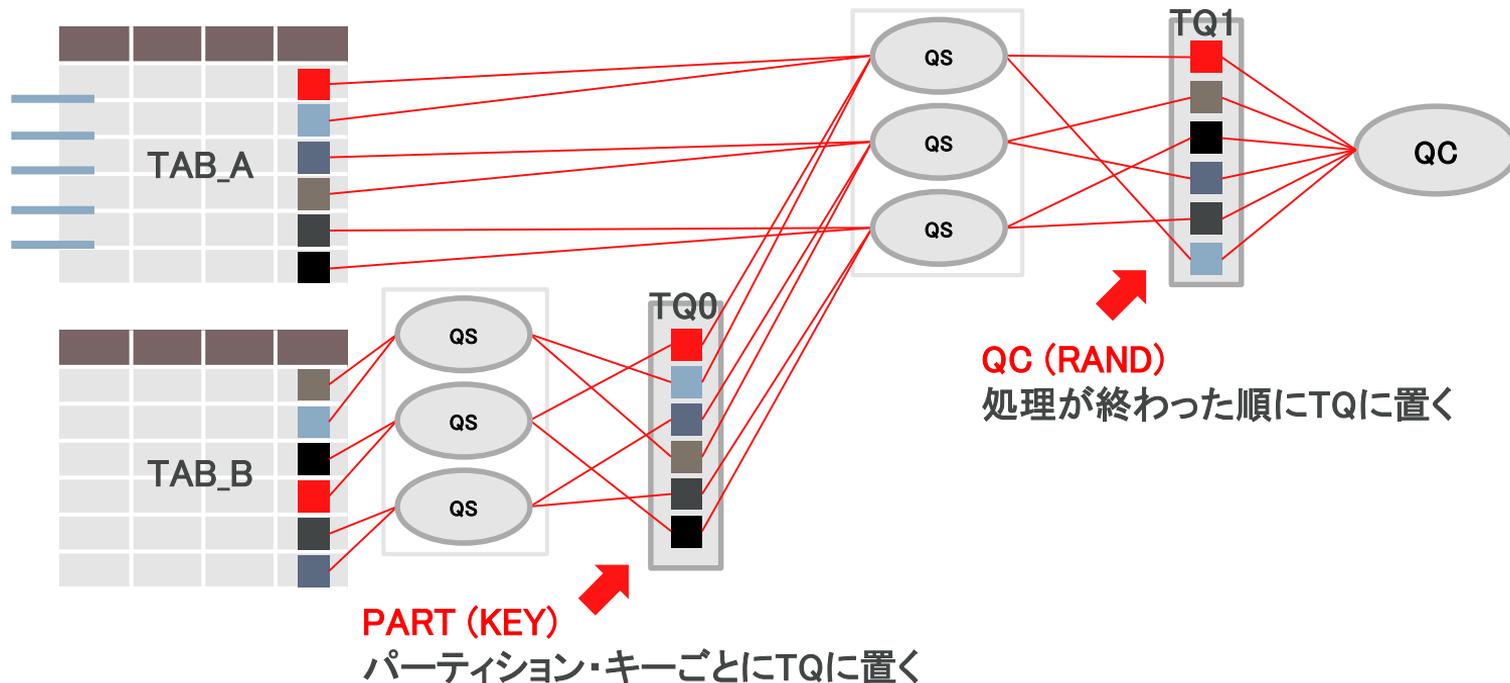
Execution Plan

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10000	Q1,00	P->S	QC (RAND)
3	PX PARTITION RANGE ALL	...	Q1,00	PCWC	
* 4	<u>HASH JOIN</u>		Q1,00	PCWP	
* 5	<u>TABLE ACCESS FULL</u>	COSTS	Q1,00	PCWP	
* 6	<u>TABLE ACCESS FULL</u>	SALES	Q1,00	PCWP	

PART (KEY) 分散方式

パーシャル・パーティション・ワイズ結合

1個の表だけが結合キーでパーティションされている場合



PART (KEY) 分散方式の実行計画例

SQL Text

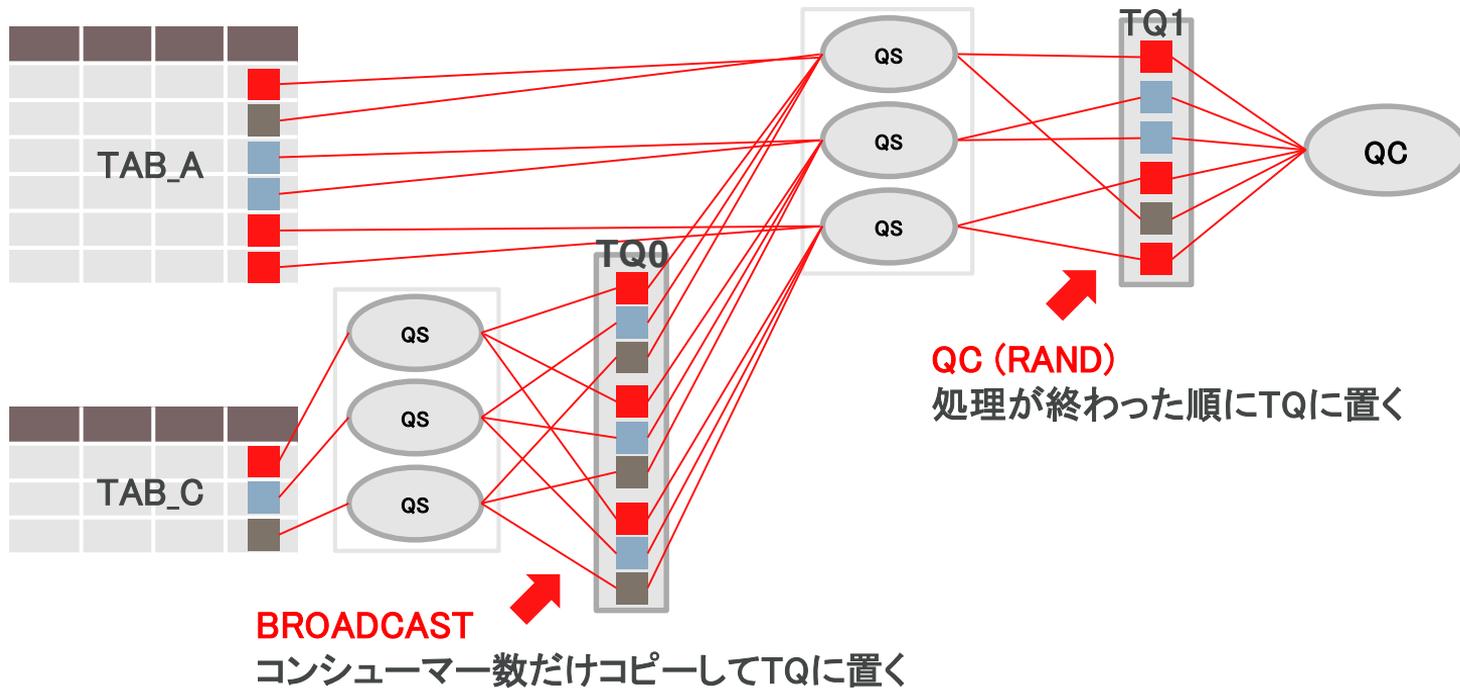
```
SELECT s.prod_id, t.day_name FROM sales s, times t WHERE s.time_id = t.time_id AND s.amount_sold >= 1000
```

Execution Plan

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10001	Q1,01	P->S	QC (RAND)
* 3	HASH JOIN		Q1,01	PCWP	
4	<u>PX RECEIVE</u>		Q1,01	PCWP	
5	<u>PX SEND PARTITION (KEY)</u>	:TQ10000 ...	Q1,00	P->P	<u>PART (KEY)</u>
6	PX BLOCK ITERATOR		Q1,00	PCWC	
7	TABLE ACCESS FULL	TIMES	Q1,00	PCWP	
8	PX PARTITION RANGE ALL		Q1,01	PCWC	
* 9	TABLE ACCESS FULL	SALES	Q1,01	PCWP	

BROADCAST分散方式

大きな表と小さな表を結合する場合



BROADCAST分散方式の実行計画例

SQL Text

```
SELECT cu.cust_email FROM customers cu, countries co
WHERE cu.country_id = co.country_id AND co.country_region = 'Asia'
```

Execution Plan

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10001	Q1,01	P->S	QC (RAND)
* 3	HASH JOIN		Q1,01	PCWP	
4	<u>PX RECEIVE</u>		Q1,01	PCWP	
5	<u>PX SEND BROADCAST</u>	:TQ10000 ...	Q1,00	P->P	<u>BROADCAST</u>
6	PX BLOCK ITERATOR		Q1,00	PCWC	
* 7	TABLE ACCESS FULL	COUNTRIES	Q1,00	PCWP	
8	PX BLOCK ITERATOR		Q1,01	PCWC	
9	TABLE ACCESS FULL	CUSTOMERS	Q1,01	PCWP	

HASHすべきなのにBROADCASTになっている

よくあるパフォーマンス問題

- 3人の顧客名簿だと思っていたのに、実は1万人の顧客名簿だった！
- 何が起きるか
 - 重い作業が分割できない
 - 大きな顧客名簿が重複コピーされるので、一次領域を大量に消費する
- なぜ起きるのか
 - 統計情報の精度が悪い

適応計画 (Adaptive Plans)

Oracle Database 12c の新機能

- 統計情報だけしか使えない実行計画作成時点では分散方法(や結合方法)を決定せず、実行時の行数を考慮して、分散方法(や結合方法)を実行時に決定する
- 行数が多ければHASH
- 行数が少なければBROADCAST
- 詳しくは<<http://www.oracle.com/technetwork/jp/ondemand/db12c-perf-1985161-ja.pdf>>

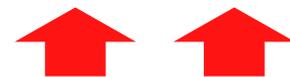
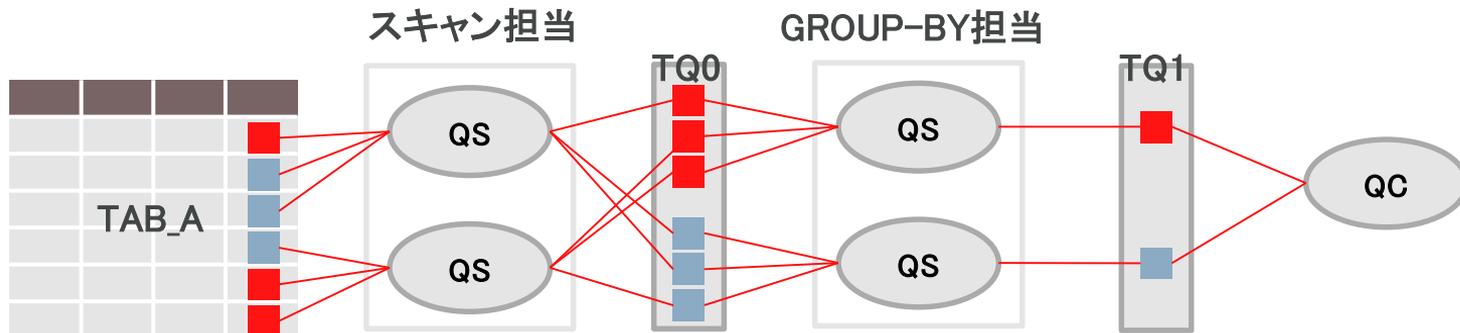
結合時の分散方式を手動で制御する

PQ_DISTRIBUTEヒント

- PQ_DISTRIBUTE(**内部表** **外部表の分散処理** **内部表の分散処理**)
- HASH HASH
 - HASH分散
- NONE NONE
 - フル・パーティション・ワイズ結合
- NONE PARTITION PARTITION NONE
 - PART (KEY) 分散
- NONE BROADCAST BROADCAST NONE
 - BROADCAST分散

パラレル実行時の GROUP-BY

シンプルなGROUP-BY



ここに注目！

6個のデータがIPCされている

シンプルなGROUP-BYの実行計画例

SQL Text

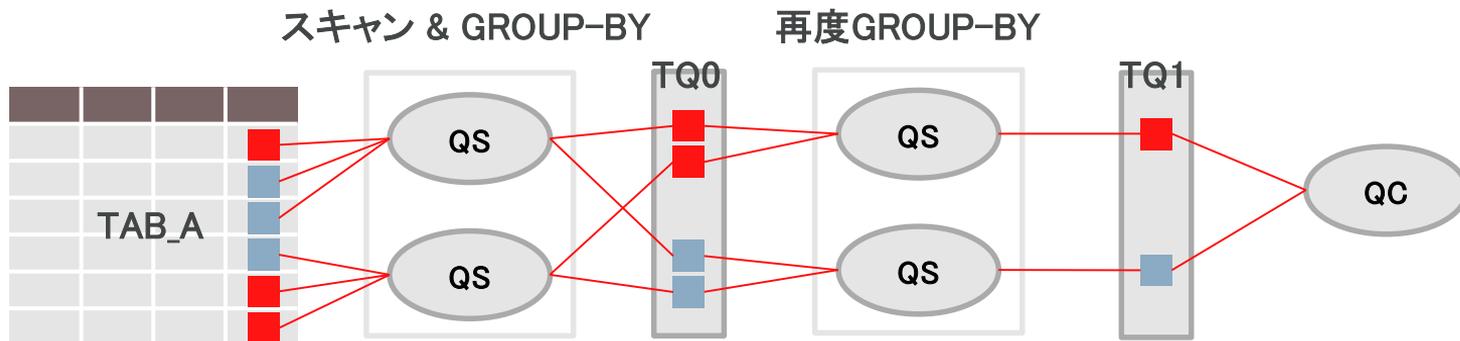
```
SELECT SUM(amount_sold) FROM sales GROUP BY prod_id
```

Execution Plan

Id	Operation	Name	Rows	Bytes	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT		72	648			
1	PX COORDINATOR						
2	PX SEND QC (RANDOM)	:TQ10001	72	648	Q1,01	P->S	QC (RAND)
3	<u>HASH GROUP BY</u>		72	648	Q1,01	PCWP	
4	<u>PX RECEIVE</u>		918K	8075K	Q1,01	PCWP	
5	<u>PX SEND HASH</u>	:TQ10000	918K	8075K	Q1,00	P->P	HASH
6	PX BLOCK ITERATOR		918K	8075K	Q1,00	PCWC	
7	TABLE ACCESS FULL	SALES	918K	8075K	Q1,00	PCWP	

GROUP-BY Pushdown

GROUP-BYを2回行なう代わりに、IPC転送量が減る



4個のデータにIPC量が減った

GROUP-BY Pushdown している実行計画例

SQL Text

```
SELECT SUM(amount_sold) FROM sales GROUP BY prod_id
```

Execution Plan

Id	Operation	Name	Rows	Bytes	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT		72	648			
1	PX COORDINATOR						
2	PX SEND QC (RANDOM)	:TQ10001	72	648	Q1,01	P->S	QC (RAND)
3	<u>HASH GROUP BY</u>		72	648	Q1,01	PCWP	
4	<u>PX RECEIVE</u>		72	648	...	PCWP	
5	<u>PX SEND HASH</u>	:TQ10000	72	648	Q1,00	P->P	HASH
6	<u>HASH GROUP BY</u>		72	648	Q1,00	PCWP	
7	PX BLOCK ITERATOR		918K	8075K	Q1,00	PCWC	
8	TABLE ACCESS FULL	SALES	918K	8075K	Q1,00	PCWP	

GROUP-BY Pushdown

- IPC量が多いコストと、2回GROUP-BYするコストを
オプティマイザが比較して自動で決定される
 - マニュアルに載っていないヒント: `GBY_PUSHDOWN` / `NO_GBY_PUSHDOWN`
- RACでインターノード・パラレル実行するときに、
インターコネクタ転送量を削減できる
- 大量のPGA要求で一時領域読み書きが多いときにも有効
- バッドノウハウ: `DISTINCT`は `GROUP-BY Pushdown` されないので
`GROUP-BY`に書き換えると速くなることがある

チューニング ケーススタディ

パラレル問合せが終わらない！

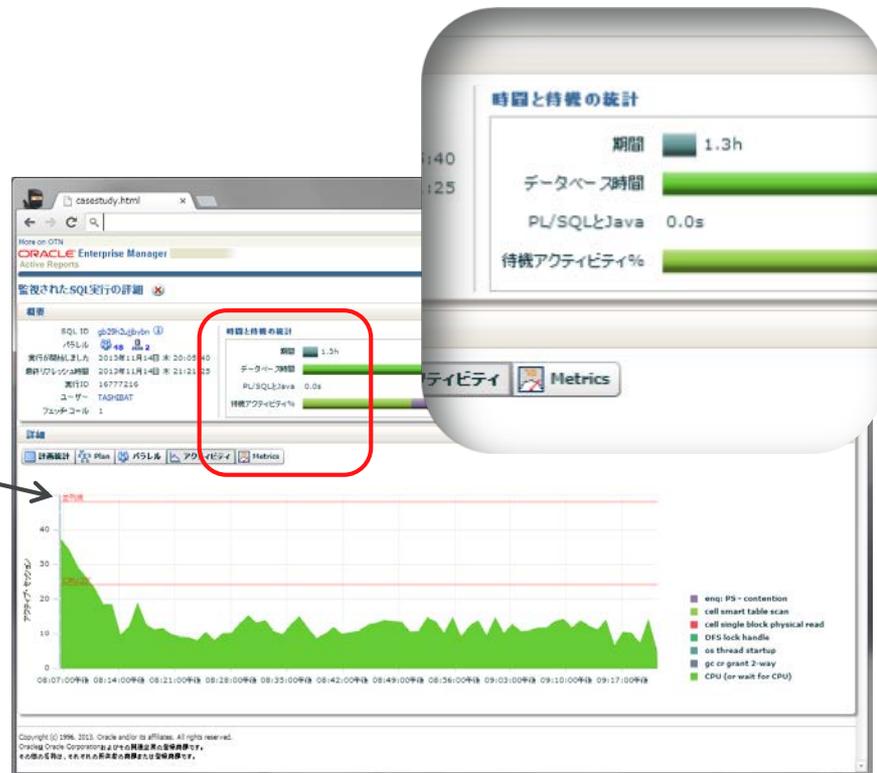
リアルタイムSQL監視のアクティビティ・タブ

- 何の待機イベントもないのに、CPUが並列度に達していない



何もしていないQSがある

並列度



リアルタイムSQL監視の平行・タブ

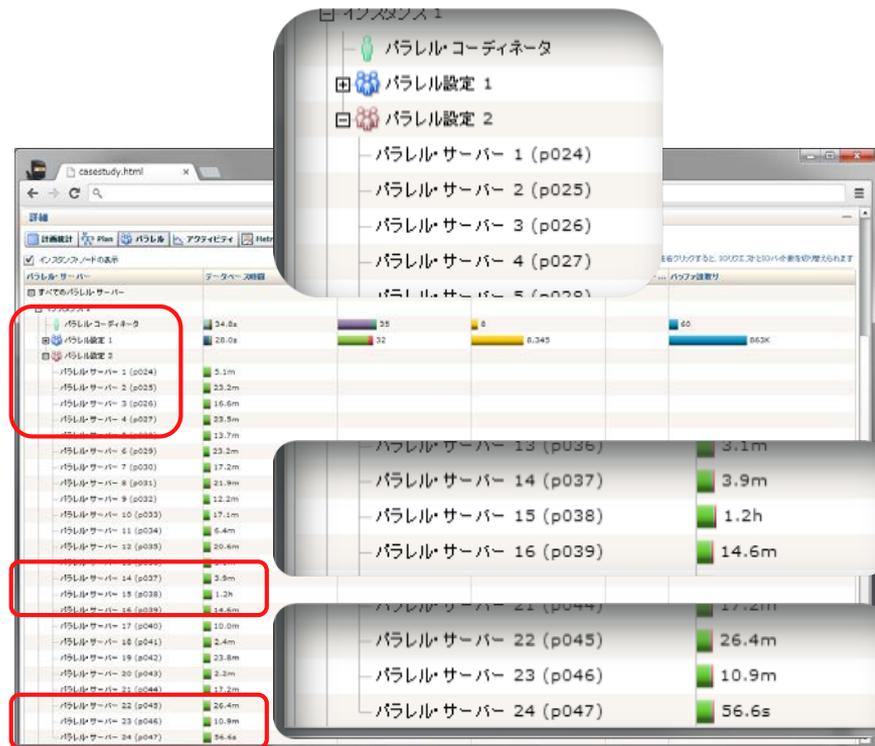
- 全体が1.3時間の状況で、56秒しか掛かっていないQSから1.2時間も掛かっているQSまでである



データの分散が偏っている

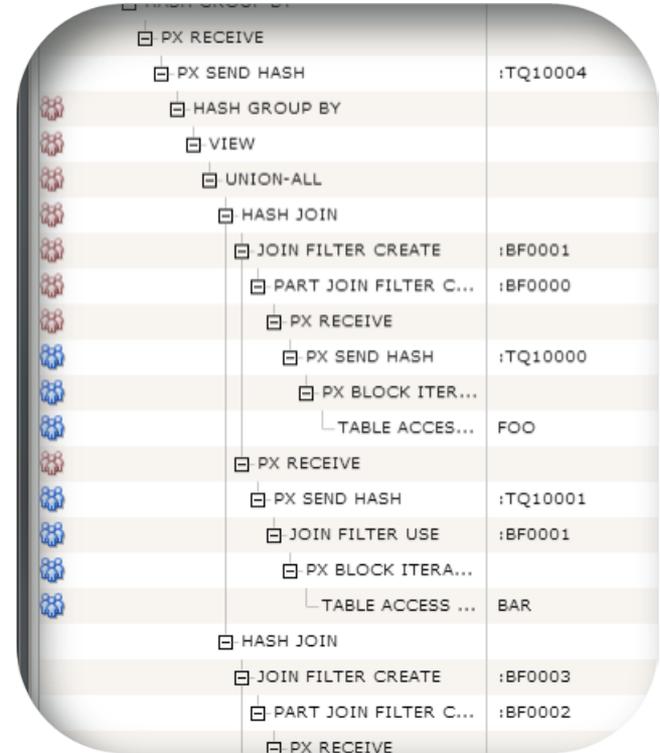


どのキーで偏っているのか？



リアルタイムSQL監視の計画統計

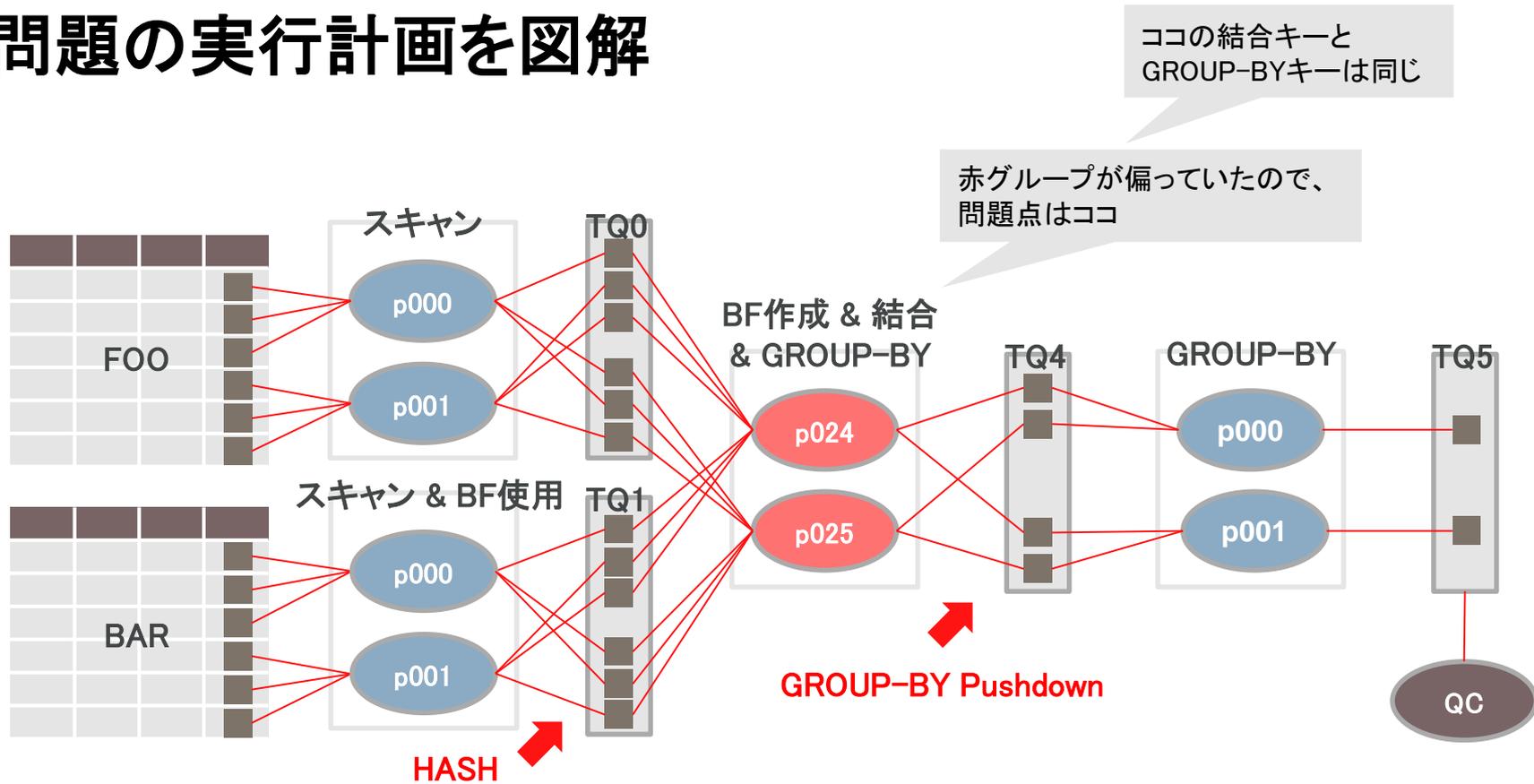
操作	名前	予測した...	コスト係数(4543)	実行	実行行	メモリ...	一時...	IOリソース	セグ...	CP/Aクエリ効率%	消費クエリ効率%
SELECT STATEMENT		0		97	0						
PX COORDINATOR		0		97	0						
PX SEND QC (RANDOM)	:TQ10003	3	67K	0	0						35
HASH GROUP BY		3	67K	0	0						
PX RECEIVE		3	67K	0	0						
PX SEND HASH	:TQ10004	3	67K	48	0						
HASH GROUP BY		3	67K	48	0						
VIEW		450	27K	40	250				23	18	
UNION-ALL		0		40	250						69
HASH JOIN		280	14K	48	250	265MB			8.66		
JOIN FILTER CREATE	:BF0001	2.099K	2.53	48	2,054K						
PART JOIN FILTER C...	:BF0000	2.099K	2.53	48	2,054K						
PX RECEIVE		2.099K	2.53	48	2,054K						
PX SEND HASH	:TQ10000	2.099K	2.53	48	1,964K						
PX BLOCK ITER...		2.099K	2.53	48	1,964K						
TABLE ACCESS...	FOO	2.099K	2.53	601	1,964K		5,600	96	0		45
PX RECEIVE		33M	9,63	48	14M						
PX SEND HASH	:TQ10001	33M	9,63	44	16M				0	.01	
JOIN FILTER USE	:BF0001	33M	9,63	44	16M						
PX BLOCK ITER...		33M	9,63	44	16M						
TABLE ACCESS...	BAR	33M	9,63	517	16M		10K	97			20
HASH JOIN		170	13K	0	0						
JOIN FILTER CREATE	:BF0003	2.099K	2.53	0	0						
PART JOIN FILTER C...	:BF0002	2.099K	2.53	0	0						
PX RECEIVE		2.099K	2.53	0	0						
PX SEND HASH	:TQ10002	2.099K	2.53	0	0						
PX BLOCK ITER...		2.099K	2.53	0	0						
TABLE ACCESS...	FOO	2.099K	2.53	0	0						
PX RECEIVE		20M	9,63	0	0						



問題の実行計画

Id	Operation	Name	TQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT				
1	PX COORDINATOR				
2	PX SEND QC (RANDOM)	:TQ10005	Q1,05	P->S	QC (RAND)
3	HASH GROUP BY		Q1,05	PCWP	
4	PX RECEIVE		Q1,05	PCWP	
5	PX SEND HASH	:TQ10004	Q1,04	P->P	HASH
6	HASH GROUP BY		Q1,04	PCWP	
7	VIEW		Q1,04	PCWC	
8	UNION ALL		Q1,04	PCWP	
9	HASH JOIN		Q1,04	PCWP	
10	JOIN FILTER CREATE	:BF0001	Q1,04	PCWP	
11	PART JOIN FILTER CREATE	:BF0000	Q1,04	PCWP	
12	PX RECIVE		Q1,04	PCWP	
13	PX SEND HASH	:TQ10000	Q1,00	P->P	HASH
14	PX BLOCK ITERATOR		Q1,00	PCWC	
15	TABLE ACCESS STORAGE FULL	FOO	Q1,00	PCWP	
16	PX RECEIVE		Q1,00	PCWP	
17	PX SEND HASH	:TQ10001	Q1,01	P->P	HASH
18	JOIN FILTER USE	:BF0001	Q1,01	PCWP	
19	PX BLOCK ITERATOR		Q1,01	PCWC	
20	TABLE ACCESS STORAGE FULL	BAR	Q1,01	PCWP	

問題の実行計画を図解



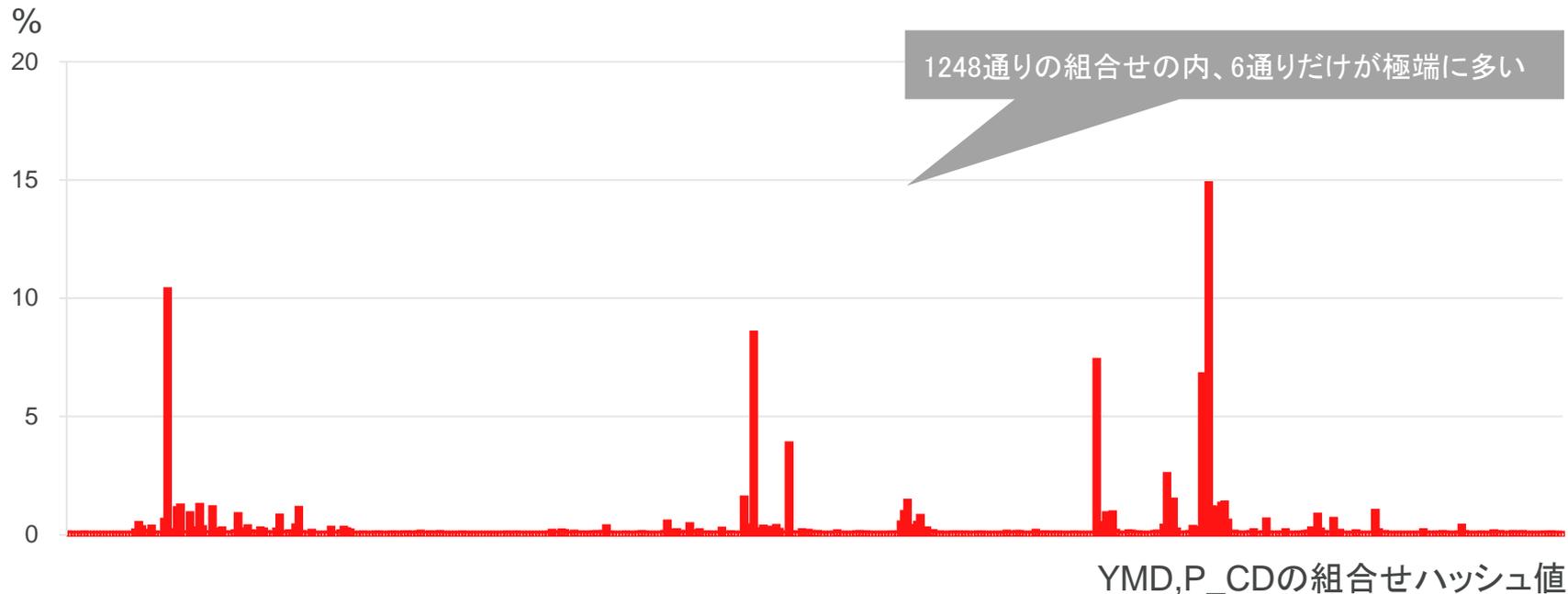
SELECT文を確認

- 結合キーはYMD, P_CD
つまり、赤グループはYMD, P_CDで
ハッシュ分割されている
- 次の青グループはYMD, S_KBNで
ハッシュ分割されている

```
SELECT ...  
FROM (  
  SELECT ...  
  FROM  
    foo f  
  INNER JOIN bar b  
  ON  
    f.ymd = b.ymd  
    AND f.p_cd = b.p_cd  
  WHERE  
    f.k_cd = 27  
    AND f.flag = 0  
    AND f.s_kbn IN (1, 2, 3)  
  UNION ALL  
  ...  
) iv  
GROUP BY iv.ymd, iv.s_kbn
```

YMD,P_CDの偏り

各YMD,P_CDの組合せが全体の何パーセントを占めるか



数量が多い上位6個のP_CDだけ別々に実施する チューニング策

```
SELECT ...
  FROM
    foo f
  INNER JOIN bar b
  ON
    f.ymd = b.ymd
  AND f.p_cd = b.p_cd
 WHERE
    f.k_cd = 27
  AND f.flag = 0
  AND f.s_kbn IN (1, 2, 3)
  AND (f.ymd, f.p_cd) IN (
    (20131101, 3), (20131104, 71), (20131104, 612),
    (20131108, 18), (20131108, 2), (20131114, 287)
  )
 UNION ALL
.....
```

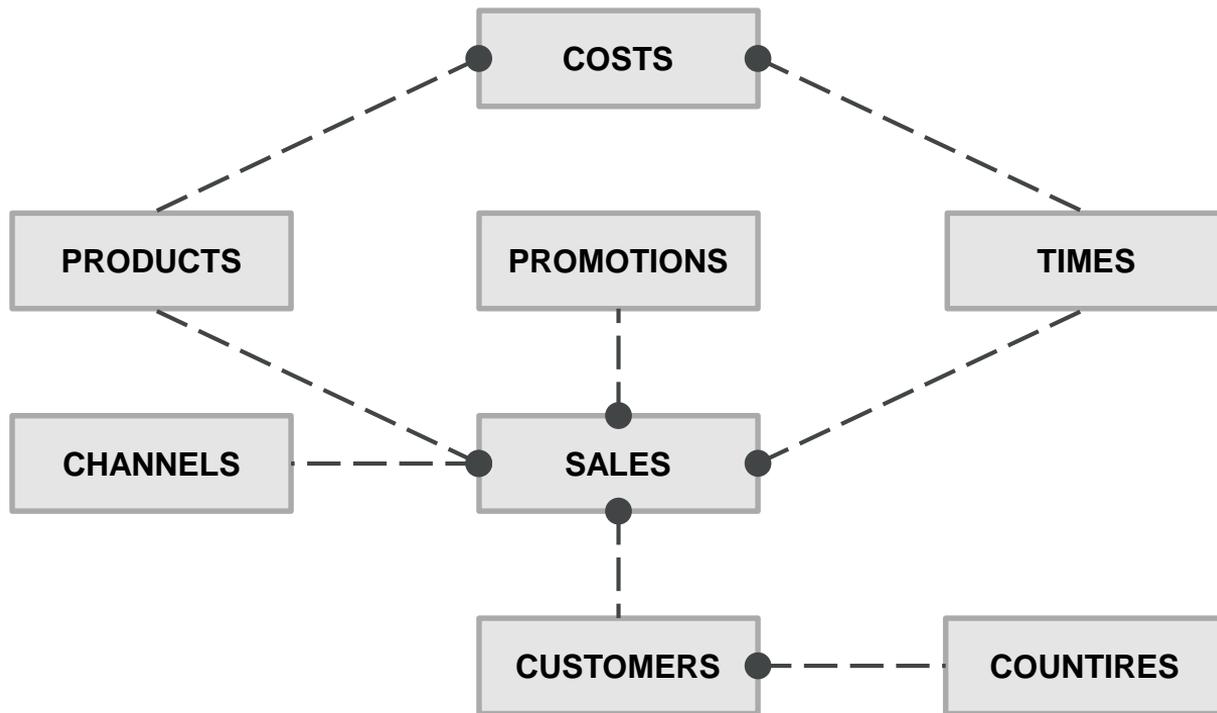
UNION
ALL

```
SELECT ...
  FROM
    foo f
  INNER JOIN bar b
  ON
    f.ymd = b.ymd
  AND f.p_cd = b.p_cd
 WHERE
    f.k_cd = 27
  AND f.flag = 0
  AND f.s_kbn IN (1, 2, 3)
  AND (f.ymd, f.p_cd) NOT IN (
    (20131101, 3), (20131104, 71), (20131104, 612),
    (20131108, 18), (20131108, 2), (20131114, 287)
  )
 UNION ALL
.....
```

まとめ

- データの流れを意識する: PQ Distrib
 - ■ の気持ちになる
- IPC(プロセス間通信)を意識する: GROUP-BY Pushdown
 - ○ の気持ちになる
- 上記をていねいに意識することで、
パラレル実行のチューニングの糸口が見えてくる

付録A: SHスキーマ図



付録B: SHスキーマの各表の行数

表名	行数	備考
CHANNELS	5	
COUNTRIES	23	
PROMOTIONS	503	
TIMES	1,826	
CUSTOMERS	55,500	
COSTS	82,112	TIME_ID列でレンジ・パーティション
SALES	918,843	TIME_ID列でレンジ・パーティション

Hardware and Software

ORACLE®

Engineered to Work Together

ORACLE®