

Oracle Database Connect 2016 DB性能の極意

シバタツ流！
私は何をどう見て
チューニングしているのか

ORACLE[®] 12^c
DATABASE

Plug into the Cloud



日本オラクル株式会社
クラウド・テクノロジー事業統括
データベースエンジニアリング本部
シニアマネジャー
柴田竜典

JPOUG

Japan Oracle User Group

ORACLE[®]

自己紹介

- データベース・パフォーマンス・チューニング・エンジニアとして13年目
- 顧客の実アプリケーションと実データを使った Proof of Concept (PoC) を Oracle Exadata で実施
- データベース基盤と管理の「それって本当？」シリーズ第1弾 データベース性能は“わんこそば”で考えよう！ 好評配布中

ORACLE®

データベース基盤と管理の「それって本当？」——
スペシャリストが真実を暴く **その1**

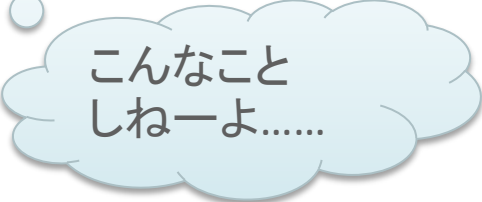
**フラッシュ・ストレージを導入すれば
CPUコア数は本当に減らせるの？
データベース性能は
“わんこそば”で考えよう！**

「フラッシュ・ストレージを導入すれば、データベースが高速化する」という話と合わせて、「フラッシュ・ストレージを導入すれば、データベース・サーバーのCPUコア数を減らせて、コスト・リットもある」という話を聞いたことがないでしょうか。「ストレージが速くなったのだから、代わりにCPUを減らせる」と言われると、なんとなくそんな気もするかもしれませんが、果たして本当なのでしょうか。



パフォーマンス・チューニングとは

- × ググって出てきた
パラメーター値をコピペ
- × 他システムでうまくいった方法を
そのまま流用
- × 社内の
言い伝えに基づいたルール



こんなこと
しねーよ.....

- どこで時間が掛かっているかを
論理的に見つけだし、
掛かっている時間を減らす

**本セッションは
どこで時間が掛かっているかを
見つけるケース・スタディ**

「今朝から急にアプリが遅い」とユーザーからクレーム

あなたはAWRやSTATSPACKの
どこから見る？

たとえば

キャッシュ・ヒット率

- キャッシュ・ヒット率がいくつであれば問題ない？ その根拠は？
- データ・ウェアハウス・システムならいくつであれば良い？
 - ファクト表が2TBある場合は？
 - ダイレクト・パス読取りはキャッシュ・ヒット率に考慮されないけどどうする？

あるAWRレポートからの引用

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	100.00
Buffer Hit %:	99.25
Library Hit %:	99.60
Execute to Parse %:	94.28
Parse CPU to Parse Elapsed %:	70.93

キャッシュ・ヒット率は十分に高く、問題ない？

Top 5 Timed Foreground Events by Total Wait Time

Event	% DB time
DB CPU	54.8
db file sequential read	33.7
utl_file I/O	3.2
gc current grant 2-way	2.0
gc cr grant 2-way	1.3

キャッシュ・ヒット率は本当に十分？

db file sequential read が多い。なるほど、その次は？

× メモリーを増設しよう

○ 他の時間帯はどうなんだろう？

○ 現在の Buffer Cache サイズは？

○ 何ブロック読んでいる？

○ どのSQLがディスクから
読んでいる？

Top n Timed

Foreground Events から
見るのはなかなか良い

でも、私は Top n より先に
見る項目があります

AWRレポート取得期間に対するデータベース時間

	Snap Id	Snap Time
Begin Snap:	71	19-Apr-15 16:45:56
End Snap:	72	19-Apr-15 17:17:12
Elapsed:		31.27 (mins)
DB Time:		8.00 (mins)

※ SQLが同時実行 / パラレル実行
されている場合、
DB Time はそれらの積上げになる

AWR取得期間に対して
データベース時間が短い



データベース以外のところ
(おそらくアプリ側)で
時間が掛かっている



データベースを分析しても
意味がない

このセッションで伝えたいこと

ここまで前振り！

- 私がAWRレポートやSQL監視レポートのどこから見ているのか — 視線の動き
- AWRレポートやSQL監視レポートにどのようなシグナルがありそれが何を意味するのか

アジェンダ

1. OLTP系のAWRレポート
2. バッチ系のAWRレポート
3. SQL監視レポート その1
4. SQL監視レポート その2
5. パフォーマンス・ハブ・アクティブ・レポート

OLTP系のAWRレポート ケース1

WORKLOAD REPOSITORY report for

DB Name	DB Id	Instance	Inst num	Startup Time	Release	RAC
	1031197718		1	18-Dec-15 00:12	11.2.0.3.0	YES

Host Name	Platform	CPUs	Cores	Sockets	Memory (GB)
	Solaris(tm) OE (64-bit)	128	16	1	255.00

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	6261	19-Dec-15 14:30:30	705	7.6
End Snap:	6265	19-Dec-15 15:30:48	701	8.4
Elapsed:		60.31 (mins)		
DB Time:		2,551.16 (mins)		

Report Summary

Cache Sizes

	Begin	End		
Buffer Cache:	79,872M	79,872M	Std Block Size:	8K
Shared Pool Size:	14,336M	14,336M	Log Buffer:	373,600K

Load Profile

	Per Second	Per Transaction	Per Exec	Per Call
DB Time(s):	42.3	0.0	0.01	0.01
DB CPU(s):	11.0	0.0	0.00	0.00
Redo size:	14,403,063.5	3,927.1		
Logical reads:	272,209.0	74.2		
Block changes:	66,264.0	18.1		
Physical reads:	11,462.0	3.1		
Physical writes:	1,816.6	0.5		
User calls:	7,619.4	2.1		
Parses:	14.4	0.0		
Hard parses:	0.2	0.0		
W/A MB processed:	72.9	0.0		
Logons:	0.7	0.0		
Executes:	7,766.3	2.1		
Rollbacks:	2,040.5	0.6		
Transactions:	3,667.6			

Instance Efficiency Percentages (Target 100%)

Buffer Nowait %:	99.76	Redo NoWait %:	100.00
Buffer Hit %:	99.82	In-memory Sort %:	100.00
Library Hit %:	100.00	Soft Parse %:	98.88

データベースに負荷は掛かっているか？

	Snap Id	Snap Time	Sessions	Cursors/Session
Begin Snap:	6261	19-Dec-15 14:30:30	705	7.6
End Snap:	6265	19-Dec-15 15:30:48	701	8.4
Elapsed:		60.31 (mins)		
DB Time:		2,551.16 (mins)		

↑
掛かっている

Top 5 Timed Foreground Events は？

Event	Waits	Time(s)	Avg wait (ms)	% DB time	Wait Class
DB CPU		39,792		26.00	
gc current block 2-way	19,337,515	19,008	1	12.42	Cluster
gc current block 3-way	8,914,254	17,705	2	11.57	Cluster
log file sync	5,862,822	13,722	2	8.96	Commit
gc cr block 2-way	11,819,932	10,249	1	6.70	Cluster

キャッシュ・フュージョンに時間が掛かっている
1回当たりは2ミリ秒以下だが、回数が多い

より道: 平均値が何百ミリ、何秒掛かっている場合

- 極端に遅い1回が平均を引き上げている(分散が大きい)のか、まんべんなく遅い(分散が小さい)のかを、以下のヒストグラムから確認すべき
 - Wait Event Histogram
 - Wait Event Histogram Detail (64 msec to 2 sec)
 - Wait Event Histogram Detail (4 sec to 2 min)
 - Wait Event Histogram Detail (4 min to 1 hr)

Segment Statistics を見てみる

Segment by Current Blocks Received

Owner	Tablespace Name	Object Name	Obj. Type	Current Block Received	%Total
USER1	TS_IND_1	TAB1_ID_1	INDEX	2,109,155	71.20

TAB1_ID_1索引への
書込み時にキャッシュ・フュージョンが多発している

念のため、原因と思われるSQLの推定

SQL ordered by Cluster Wait Time

Cluster Wait Time (s)	Executions	Elapsed Time (s)	SQL Text
22,526.77	58,855	29,108.95	INSERT INTO tab1 VALUES (tab1_seq.nextval, ...
7,748.50	2,235,057	9,513.76	UPDATE tab2...
5,588.78	61,709	6,739.12	SELECT col1, col2...

TAB1表のID列に
TAB1_SEQシーケンスのNEXTVALをINSERTしている

結論: TAB1_ID_1索引への書込み集中

Right Growing Index 対策

- 逆キー索引を使用する
 - ただし、リーフ・ブロックが多くなりすぎ
キャッシュ・ヒット率が悪くなるかも
- 書込みが分散するであろうキーで
ハッシュ・パーティションを作り、
ローカル索引にする
 - ただし、キャッシュ・フュージョンを
ゼロにはできない

- ```
INSERT INTO tab1 (id, ...)
VALUES (
 1E+16 * SYS_CONTEXT('USERENV', 'INSTANCE') +
 1E+14 *
 MOD(SYS_CONTEXT('USERENV', 'SID'), 100) +
 tab1_seq.NEXTVAL, ...
);
```

  - インスタンス番号を含めることで  
キャッシュ・フュージョンがなくなる
  - セッションIDからの2桁を含めることで  
適度にリーフ・ブロックが分散する

# バッチ系のAWRレポート ケース2

**WORKLOAD REPOSITORY report for**

| DB Name | DB Id      | Instance | Inst num | Startup Time    | Release    | RAC |
|---------|------------|----------|----------|-----------------|------------|-----|
|         | 1796207473 |          | 1        | 03-Sep-13 17:09 | 11.2.0.3.0 | YES |

| Host Name | Platform         | CPUs | Cores | Sockets | Memory (GB) |
|-----------|------------------|------|-------|---------|-------------|
|           | Linux x86_64-bit | 16   | 8     | 2       | 252.28      |

|             | Snap Id | Snap Time          | Sessions | Cursors/Session |
|-------------|---------|--------------------|----------|-----------------|
| Begin Snap: | 178     | 03-Sep-13 18:06:35 | 53       | 1.4             |
| End Snap:   | 179     | 03-Sep-13 18:18:41 | 61       | 1.5             |
| Elapsed:    |         | 12.10 (mins)       |          |                 |
| DB Time:    |         | 12.35 (mins)       |          |                 |

**Report Summary**

Cache Sizes

|                   | Begin   | End     |                 |          |
|-------------------|---------|---------|-----------------|----------|
| Buffer Cache:     | 21,632M | 21,632M | Std Block Size: | 8K       |
| Shared Pool Size: | 2,560M  | 2,560M  | Log Buffer:     | 147,600K |

Load Profile

|                   | Per Second  | Per Transaction | Per Exec | Per Call |
|-------------------|-------------|-----------------|----------|----------|
| DB Time(s):       | 1.0         | 21.2            | 0.00     | 0.25     |
| DB CPU(s):        | 1.0         | 20.1            | 0.00     | 0.23     |
| Redo size:        | 1,486,714.0 | 30,851,056.9    |          |          |
| Logical reads:    | 6,566.6     | 136,264.3       |          |          |
| Block changes:    | 6,041.1     | 125,360.5       |          |          |
| Physical reads:   | 20.2        | 418.8           |          |          |
| Physical writes:  | 67.9        | 1,409.2         |          |          |
| User calls:       | 4.1         | 85.5            |          |          |
| Parses:           | 6.1         | 127.3           |          |          |
| Hard parses:      | 0.2         | 5.0             |          |          |
| W/A MB processed: | 0.7         | 14.7            |          |          |
| Logons:           | 0.4         | 7.8             |          |          |
| Executes:         | 4,014.6     | 83,307.1        |          |          |
| Rollbacks:        | 0.0         | 0.0             |          |          |
| Transactions:     | 0.1         |                 |          |          |

Instance Efficiency Percentages (Target 100%)

|                  |        |                   |        |
|------------------|--------|-------------------|--------|
| Buffer Nowait %: | 100.00 | Redo NoWait %:    | 100.00 |
| Buffer Hit %:    | 99.69  | In-memory Sort %: | 100.00 |
| Library Hit %:   | 99.98  | Soft Parse %:     | 96.05  |
| ...              | ...    | ...               | ...    |

# データベースに負荷は掛かっているか？

|             | Snap Id | Snap Time          | Sessions | Cursors/Session |
|-------------|---------|--------------------|----------|-----------------|
| Begin Snap: | 178     | 03-Sep-15 18:06:35 | 53       | 1.4             |
| End Snap:   | 179     | 03-Sep-15 18:18:41 | 61       | 1.5             |
| Elapsed:    |         | 12.10 (mins)       |          |                 |
| DB Time:    |         | 12.35 (mins)       |          |                 |

各ジョブの直前にAWRのスナップショットを作成しているため  
このAWRに含まれている主要なSQLは確実に1個

# Top 5 Timed Foreground Events は？

| Event                              | Waits   | Time(s) | Avg wait (ms) | % DB time | Wait Class  |
|------------------------------------|---------|---------|---------------|-----------|-------------|
| DB CPU                             |         | 702     |               | 94.75     |             |
| row cache lock                     | 970,742 | 37      | 0             | 5.03      | Concurrency |
| reliable message                   | 3,966   | 1       | 0             | 0.17      | Other       |
| cell multiblock<br>physical read   | 174     | 1       | 4             | 0.10      | User I/O    |
| cell single block<br>physical read | 414     | 0       | 1             | 0.05      | User I/O    |

row cache lock (シーケンスがキャッシュから採番できなかったときの待機イベント) の割合は5%程度だが、回数が97万回と非常に多い。SQL1個しかないのに、なぜ？

# SQL ordered by Elapsed Time を見てみる

| Elapsed Time (s) | Executions | SQL Module       | SQL Text                                         |
|------------------|------------|------------------|--------------------------------------------------|
| 716.04           | 1          | SQL*Plus         | INSERT INTO tab1 ... SELECT ...                  |
| 424.33           | 970,000    | SQL*Plus         | SELECT ...                                       |
| 214.25           | 970,000    | SQL*Plus         | SELECT tab1_seq.NEXTVAL ...                      |
| 69.15            | 970,000    | SQL*Plus         | update seq\$ set increment\$=:2, minvalue=:3 ... |
| 17.18            | 22         | SQL*Plus         | WITH view1 AS (SELECT ...                        |
| 8.43             | 21         | Admin Connection | select /*+ no_monitor */ dbms_...                |

バッチの主要SQLは  
確かに1回

97万回実行されている  
これらはなに？

シーケンスから  
97万回採番してる

シーケンスのディクショナリを  
97万回UPDATEしている

結論: INSERT先の表に行トリガーがあり、  
行トリガーがキャッシュされていない  
シーケンスから採番している

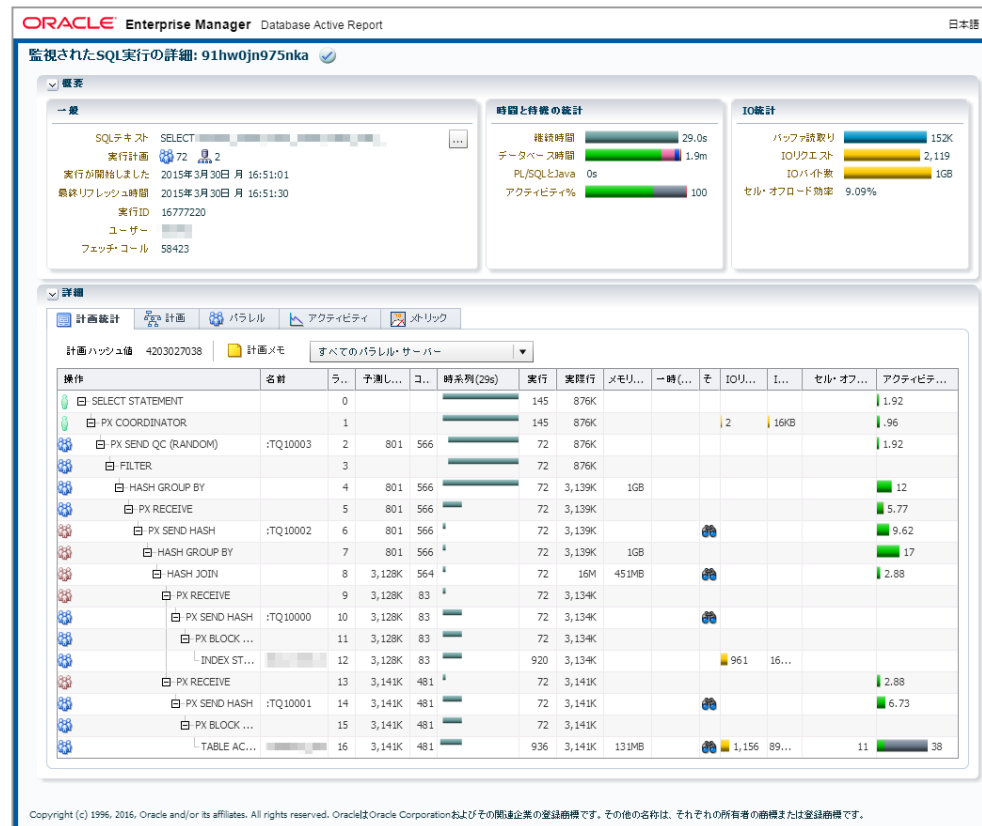
- 対策
- シーケンスにキャッシュ(できれば970,000以上)を設定する
- そもそもシーケンスから新たに採番する必要性が  
本当にあるのか再検討
  - 714.69秒の INSERT SELECT のうち、  
トリガーに要している時間が709.24秒
  - トリガーを外して実測すると、6.8秒で INSERT SELECT が完了

# シーケンスのINSERT先に 主キー制約または一意制約がある場合

- Right Growing Index 対策
- INSERT SELECT 前に  
ALTER TABLE ... MODIFY CONSTRAINT ... DISABLE NOVALIDATE しておき、  
INSERT SELECT 後に ALTER TABLE ... MODIFY CONSTRAINT ... ENABLE VALIDATE  
をパラレルDDLすることも検討すべき
  - INSERT前から入っている行数に対してINSERTする行数が多い場合は効果的
  - ALTER TABLE ... EXCHANGE PARTITION できれば、より効率的
- 索引がなくても、運用で一意性が保証できる場合
  - ALTER TABLE ... ADD CONSTRAINT ... PRIMARY KEY (...) RELY -- 索引を作らない
  - ALTER SYSTEM SET QUERY\_REWRITE\_INTEGRITY = TRUSTED -- 信じてクエリー・リライトする

# SQL監視レポート その1

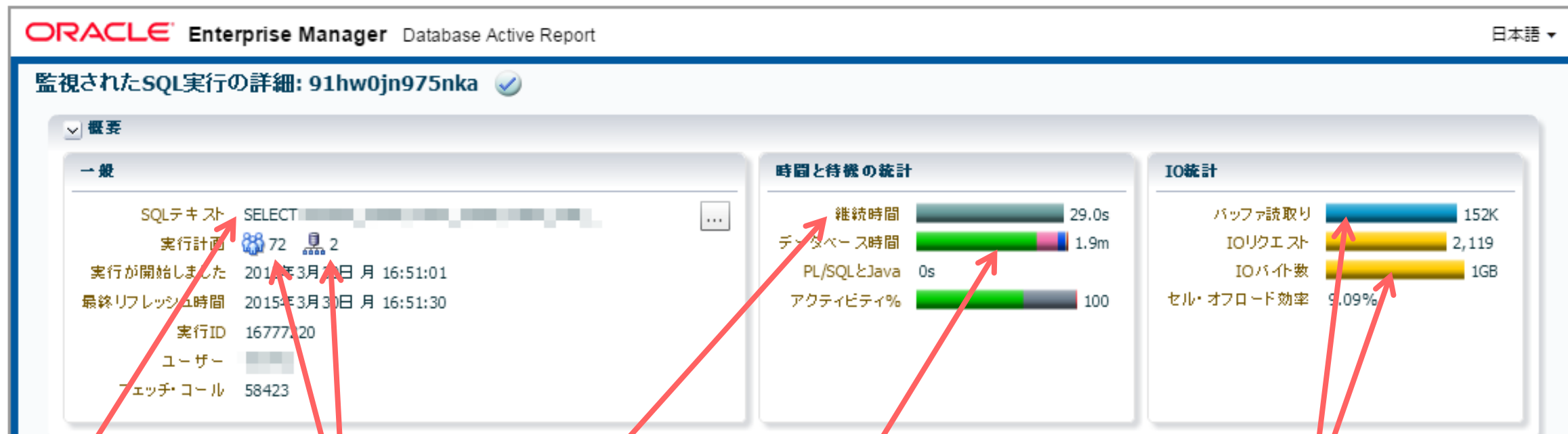
## ケース3



# リアルタイムSQL監視レポートとは

- 特定のSQLの実行中にパフォーマンスの分析が行える
- 特定のSQLの実行後にパフォーマンスの分析が行える
- テキスト・モードは11.1、アクティブHTMLモードは11.2から
- Oracle Enterprise Manager Diagnostics Pack と Tuning Pack が必要
- 取得できるSQL
  - 5秒以上時間の掛かったSQL
  - パラレル実行されたSQL
  - MONITORヒントのあるSQL
- 取得方法
  - Enterprise Manager
  - DBMS\_SQLTUNE.REPORT\_SQL\_MONITOR()

# まずは上半分を見てみよう



調査対象のSQL

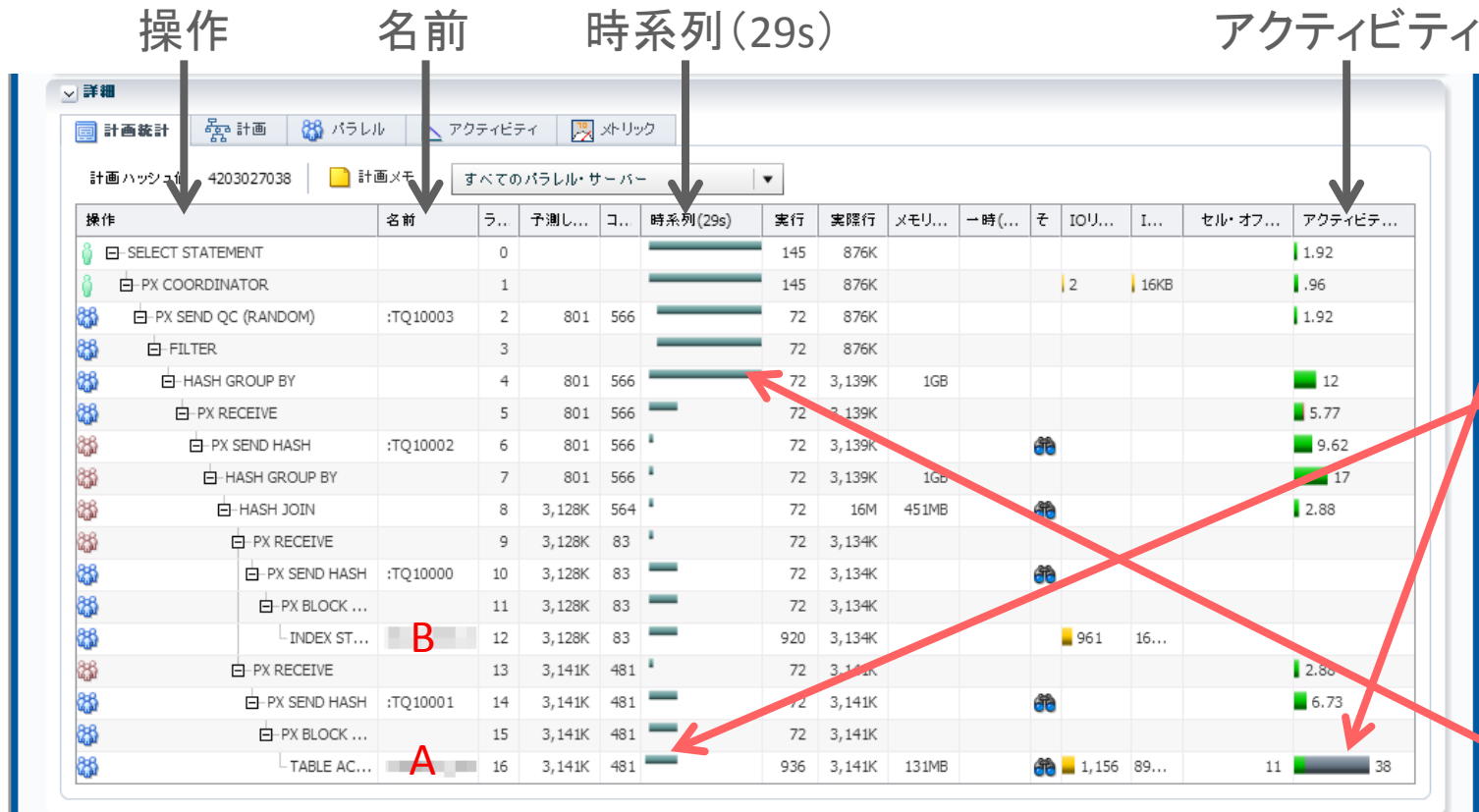
TATは29秒

CPU(緑)ネック

正しくパラレル実行されている  
2ノード上で72プロセス

ほとんどダイレクト・パス読取りしている  
IOバイト数上で平均IOサイズがポップアップ

# 計画統計タブ(下半分)を見てみよう

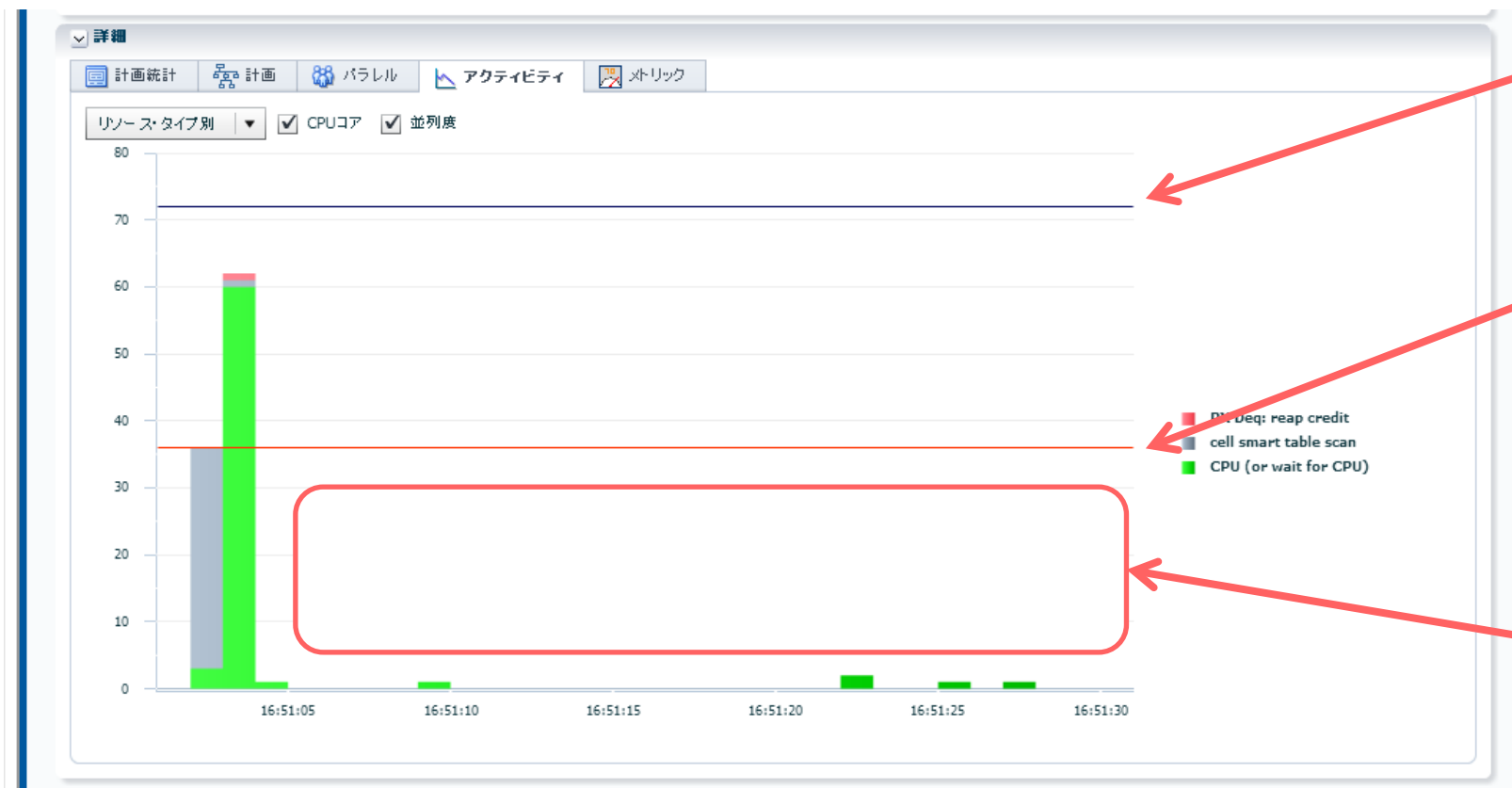


表Aの TABLE ACCESS STORAGE FULL が重い。中でも cell smart table scan 待機イベントがアクティビティ全体の33%。

ただし、表Aの TABLE ACCESS STORAGE FULL の所要時間は8秒

この HASH GROUP BY 直前までは8秒で終わっている。ここから残り21秒。軽いのに時間が掛かっている

# アクティビティ・タブを見てみよう



パラレル度  
縦棒がここまでなければ、  
アイドルなプロセスがある

物理コア数  
縦棒がここまでなければ、  
アイドルな物理コアがある

開始4秒後から、  
データベースは  
ほとんどアイドル

# RDBMSの先でつまっている？

- たとえばJavaだと、`ResultSet rset = pstmt.executeQuery()`したタイミングで、結果セットが完成しているように誤解しやすい
  - 巨大な結果セットを返すとき、それを置いておけるメモリー領域なんてない
- 実際には `rset.next()` で(フェッチ・フェーズで)フェッチ・サイズに従って、少しずつRDBMS側もバケツ・リレーしている
- SQL\*Plusは(開発ツールなので)遅い！  
結果セットが1,000行以上の場合には注意
  - Exadataでも、77,000行のDBA\_OBJECTS表の出力に15分！（1000行出力に約10秒）

# 結論: SQL\*Plusボトルネック

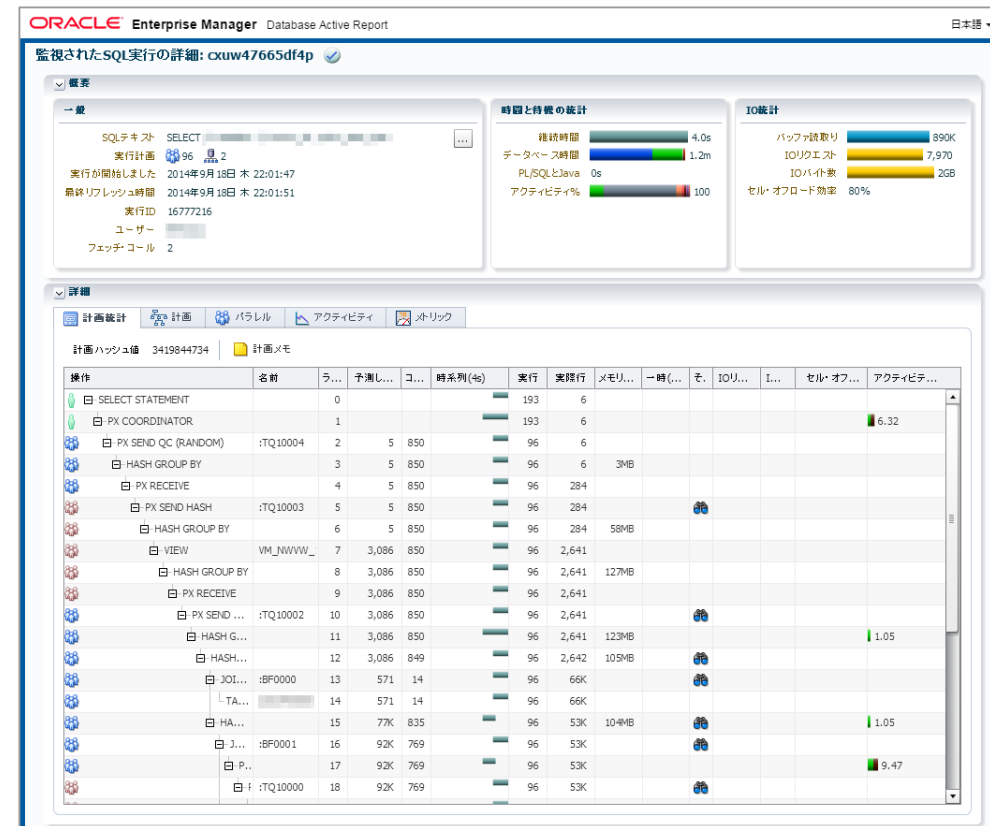
## 対策

- 本番アプリの動きを再確認
  - ページングしてるかも.....
    - していないときにROWNUMを使うと実行計画が変わる可能性があるので注意
- INSERT SELECT に変更し、クライアントの影響をひとまず無視する
  - 他社RDBMSと比較している場合など

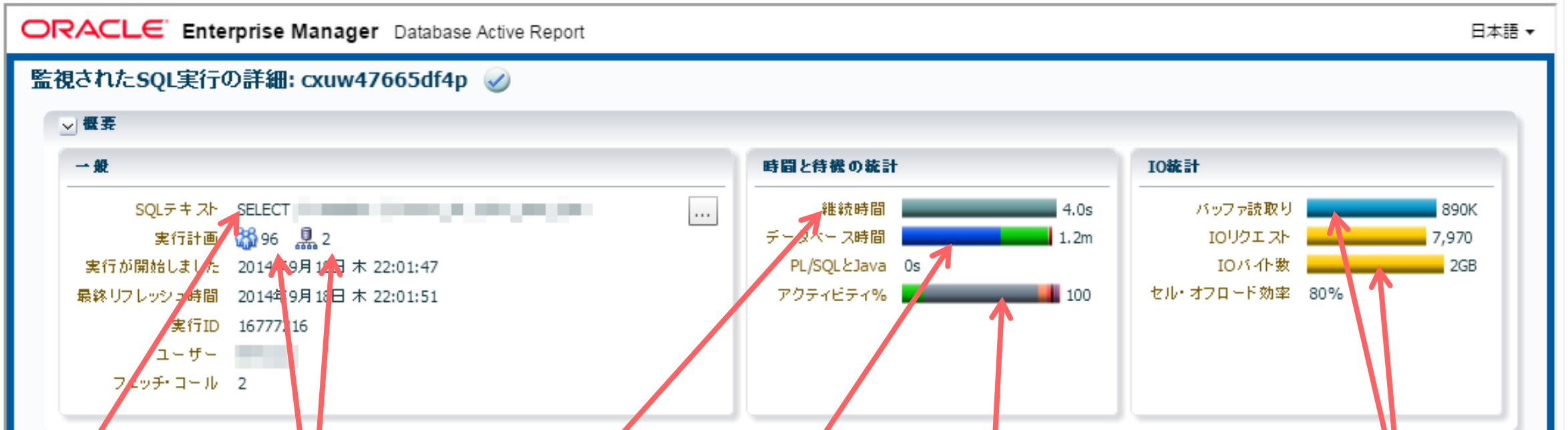
- SET AUTOTRACE TRACEONLY  
クライアントは結果をフェッチするが画面描画しない
- SET ARRAYSIZE  
フェッチ・サイズを大きくする  
“SQL\*Plusおよび Oracle Database の最近のバージョンでは、ARRAYSIZEによる効果はほとんどありません” SQL\*Plus ユーザーズ・ガイド  
およびリファレンス リリース12.1

# SQL監視レポート その2

## ケース4



# まずは上半分を見てみよう



調査対象のSQL

正しくパラレル実行

TATは4秒

ややIO(青)ネック

Concurrency: cursor: pin S wait on X

ダイレクト・パス読取り

# 計画統計タブ(下半分)を見てみよう

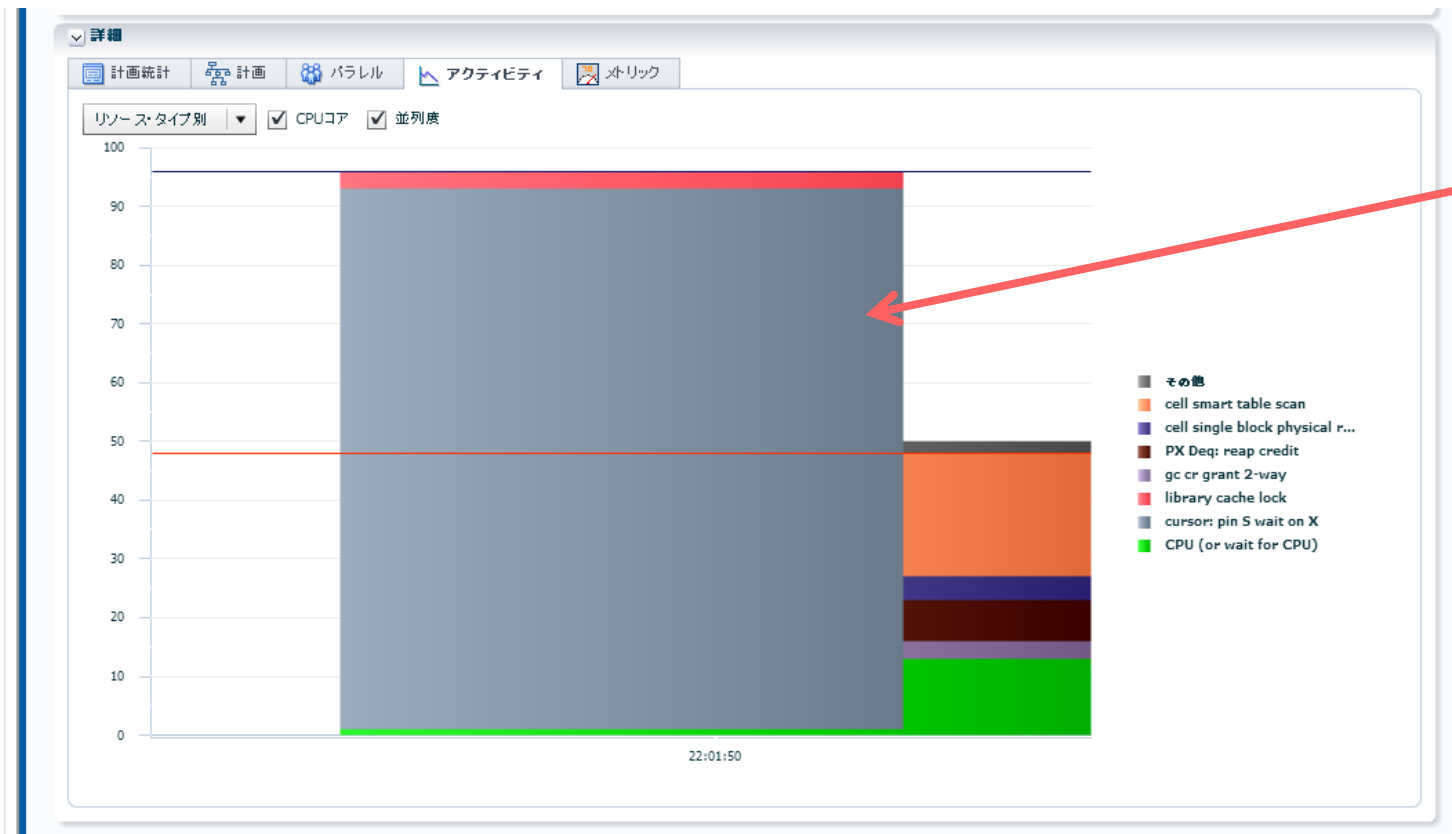


表Aの DB CPU が重い

表Bの cell table smart scan も重い

最初の5分の3くらいの空白はなに？

# アクティビティ・タブを見てみよう



最初の5分の3の時間は  
ほぼ全プロセスが  
cursor: pin S wait on X  
で待機している

# cursor: pin S wait on X とは

- cursor: pin S wait on X とは library cache lock のようなもの
  - 10g R2 からライブラリ・キャッシュの管理方法が変わったため、異なる名称の待機イベントになった
- SQL実行前に cursor: pin S wait on X が数秒発生する場合、パース関連が怪しい
  - 同時実行SQL数が少ない場合 → ハード・パースが怪しい
  - 同時実行SQL数が多い場合  
→ ハード・パースだけでなく、ソフト・パース同士が競合しているかも

# 結論: パース

## ハード・パースの対策

- 我慢する
  - 間違った実行計画が一番時間が掛かるので、パースに多少時間が掛かってもより正しい実行計画を作ろうとするのが最近のオプティマイザの傾向
- 動的統計(動的サンプリング)のレベルを下げる

## ソフト・パースの対策

- アプリケーション側で不要なprepareを減らす
- アプリケーション側で事前にprepareを終わらせておく

# パフォーマンス・ハブ・アクティブ・レポート ケース5



# パフォーマンス・ハブ・アクティブ・レポートとは

- 特定期間に流れたSQLすべてのパフォーマンスの分析が行える
  - 直近1時間のリアルタイム・モード
  - 履歴モード
- 12.1から使用可能
- Oracle Enterprise Manager Diagnostics Pack が必要
  - SQL監視部分は Tuning Pack も必要

## • 取得方法

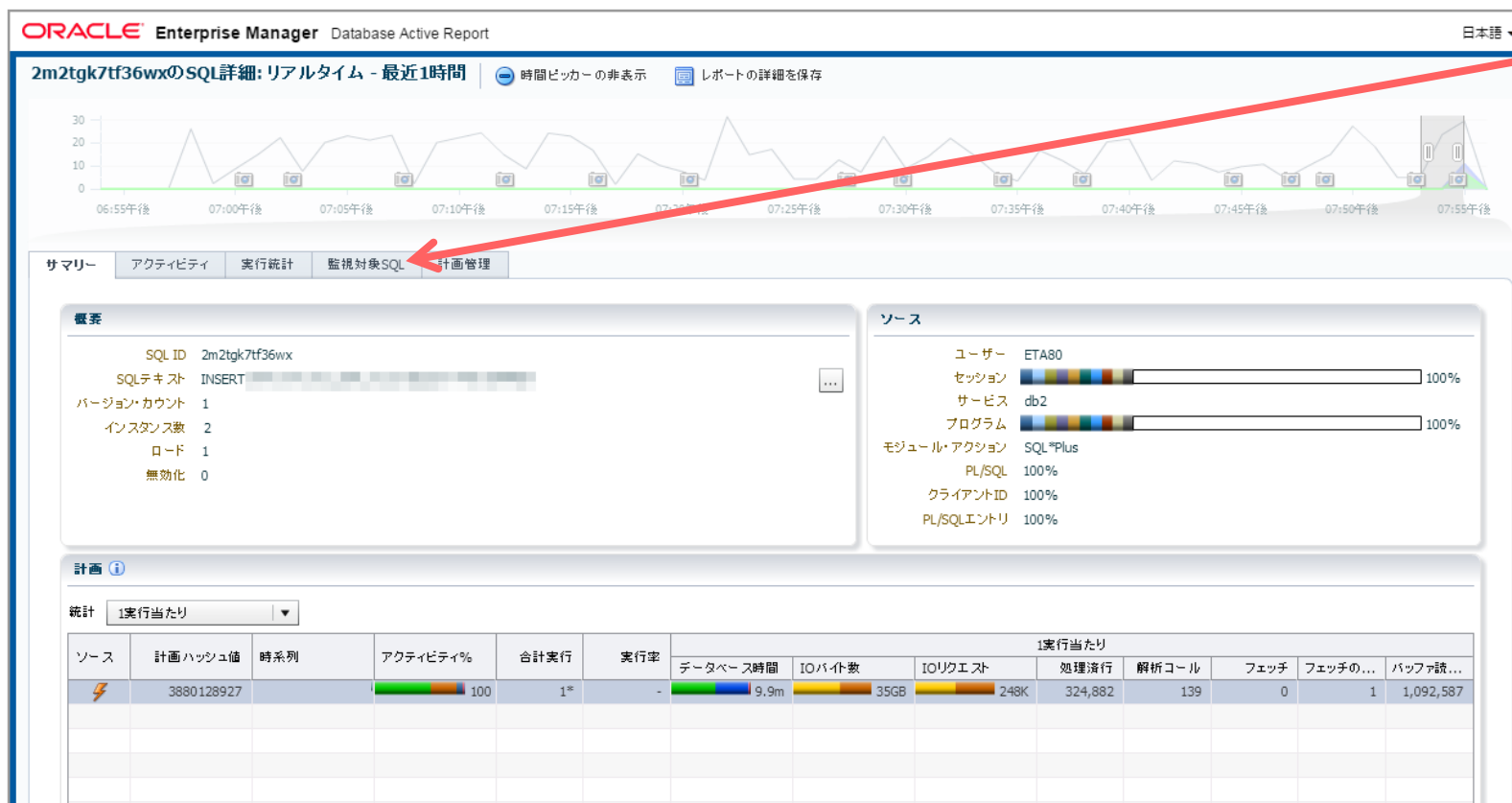
- EM Express
- @\$ORACLE\_HOME/rdbms/admin/perfhubrpt.sql
- DEFINE report\_type='active-html'  
@\$ORACLE\_HOME/rdbms/admin/awrrpt.sql
  - AWRLレポートの最後にパフォーマンス・ハブがついてくる

# リアルタイム・モードのワークロード・タブ



上位1位SQLの  
2m2tgk7tf36wxを  
クリック

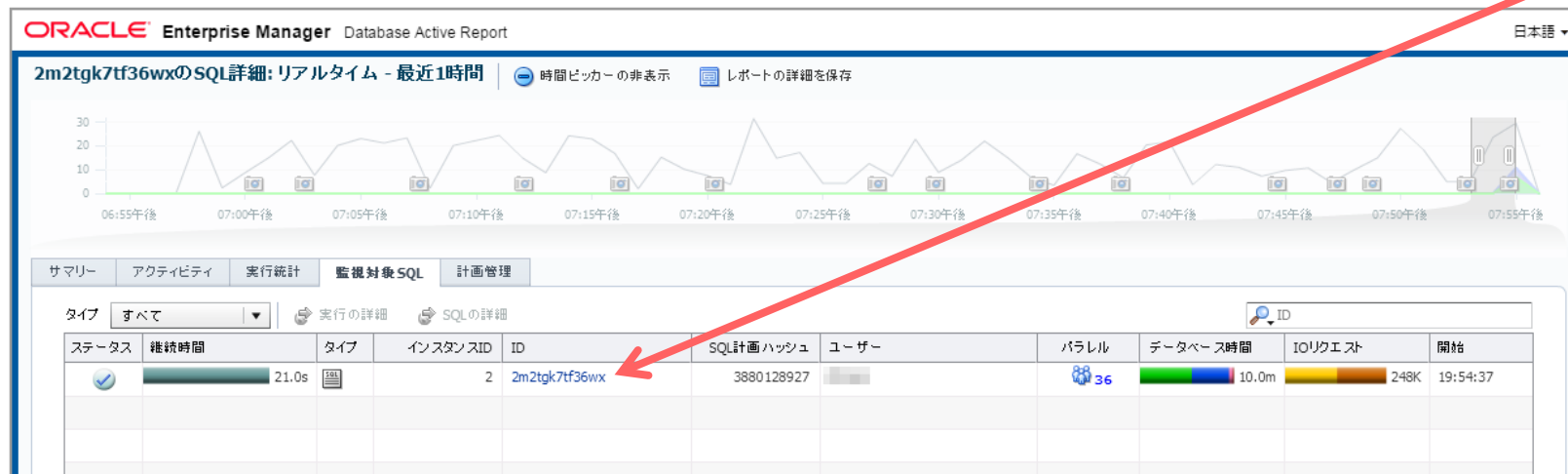
# 2m2tgk7tf36wxのSQL詳細



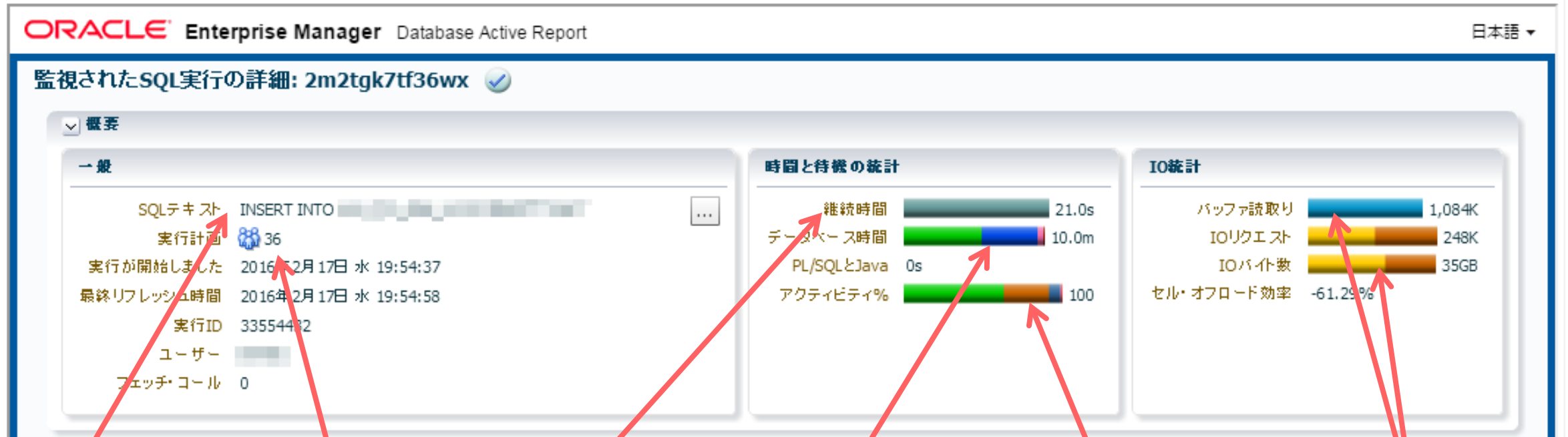
監視対象SQLをクリック

# 2m2tgk7tf35wxの監視対象SQL

2m2tgk7tf25wxを  
クリック



# SQL監視レポートが表示される



調査対象のSQL

TATは21秒  
正しくパラレル実行されている

CPU(緑)と  
IO(青)が半々

User I/O: direct path write temp

ダイレクト・パス読取り

# 計画統計タブ(下半分)を見てみよう

| 操作                            | 名前       | ライ... | 予測した行数 | コスト   | 時系列(21s) | 実行  | 実行時間   | メモリ... | 一時(...) | そ... | IOリクエスト | IOバイト数 | セル・オ... | アクティビティ% |
|-------------------------------|----------|-------|--------|-------|----------|-----|--------|--------|---------|------|---------|--------|---------|----------|
| [-] PX SEND HASH              | :TQ30003 |       | 21     | 7,883 |          | 36  | 624K   |        |         |      |         |        |         |          |
| [-] HASH JOIN SEMI BUFFERED   |          |       | 22     | 7,883 |          | 36  | 624K   | 1GB    |         |      |         |        |         | .46      |
| [-] JOIN FILTER CREATE        | :BF0001  |       | 23     | 128K  | 7,853    | 36  | 8,546K |        |         |      |         |        |         |          |
| [-] PX RECEIVE                |          |       | 24     | 128K  | 7,853    | 36  | 8,546K |        |         |      |         |        |         |          |
| [-] PX SEND HASH              | :TQ30001 |       | 25     | 128K  | 7,853    | 36  | 8,546K |        |         |      |         |        |         | .92      |
| [-] HASH JOIN                 |          |       | 26     | 128K  | 7,853    | 36  | 8,546K | 17GB   | 14GB    |      | 239K    | 27GB   |         | 70       |
| [-] JOIN FILTER CREATE        | :BF0002  |       | 27     | 126K  | 3,777    | 36  | 309M   |        |         |      |         |        |         | .11      |
| [-] PX RECEIVE                |          |       | 28     | 126K  | 3,777    | 36  | 309M   |        |         |      |         |        |         | 5.85     |
| [-] PX SEND BROADCAST         | :TQ30000 |       | 29     | 126K  | 3,777    | 36  | 309M   |        |         |      |         |        |         | 5.23     |
| [-] PX BLOCK ITERATOR         |          |       | 30     | 126K  | 3,777    | 36  | 8,586K |        |         |      |         |        |         |          |
| [-] TABLE ACCESS STORAGE FULL |          | C     | 31     | 126K  | 3,777    | 503 | 8,586K | 107MB  |         |      | 4,024   | 3GB    | 91      | .77      |
| [-] PX COORDINATOR            |          |       | 32     |       |          | 37  | 1      |        |         |      |         |        |         |          |
| [-] PX SEND QC (RANDOM)       | :TQ10000 |       | 33     | 1     | 9        | 36  | 1      |        |         |      |         |        |         |          |
| [-] PX BLOCK ITERATOR         |          |       | 34     | 1     | 9        | 36  | 1      |        |         |      |         |        |         |          |
| [-] TABLE ACCESS STORA...     |          | B     | 35     | 1     | 9        | 1   | 1      |        |         |      |         |        |         |          |
| [-] PX COORDINATOR            |          |       | 36     |       |          | 37  | 1      |        |         |      |         |        |         |          |
| [-] PX SEND QC (RANDOM)       | :TQ20000 |       | 37     | 1     | 9        | 36  | 1      |        |         |      |         |        |         |          |
| [-] PX BLOCK ITERATOR         |          |       | 38     | 1     | 9        | 36  | 1      |        |         |      |         |        |         |          |
| [-] TABLE ACCESS STORA...     |          | A     | 39     | 1     | 9        | 1   | 1      |        |         |      |         |        |         |          |
| [-] JOIN FILTER USE           | :BF0002  |       | 40     | 16M   | 4,065    | 36  | 14M    |        |         |      |         |        |         |          |

35GBのIOのうち、  
27GBがこの一時表領域IO

アクティビティの70%がここ

21秒のうち、  
17秒が表Cに対する  
最初の処理

# 表Cの処理をもう少し拡大

レベル: アクティビティ | メトリック

計画メモ: すべてのパラレル・サーバー

|                               | 名前       | ライ... | 予測した行数 | コスト   | 時系列 (21s) | 実行  | 実際行    | × |
|-------------------------------|----------|-------|--------|-------|-----------|-----|--------|---|
| [-] PX SEND HASH              | :TQ30003 | 21    | 7      | 7,883 |           | 36  | 624K   |   |
| [-] HASH JOIN SEMI BUFFERED   |          | 22    | 7      | 7,883 |           | 36  | 624K   |   |
| [-] JOIN FILTER CREATE        | :BF0001  | 23    | 128K   | 7,853 |           | 36  | 8,546K |   |
| [-] PX RECEIVE                |          | 24    | 128K   | 7,853 |           | 36  | 8,546K |   |
| [-] PX SEND HASH              | :TQ30001 | 25    | 128K   | 7,853 |           | 36  | 8,546K |   |
| [-] HASH JOIN                 |          | 26    | 128K   | 7,853 |           | 36  | 8,546K |   |
| [-] JOIN FILTER CREATE        | :BF0002  | 27    | 126K   | 3,787 |           | 36  | 309M   |   |
| [-] PX RECEIVE                |          | 28    | 126K   | 3,787 |           | 36  | 309M   |   |
| [-] PX SEND BROADCAST         | :TQ30000 | 29    | 126K   | 3,787 |           | 36  | 309M   |   |
| [-] PX BLOCK ITERATOR         |          | 30    | 126K   | 3,787 |           | 36  | 8,546K |   |
| [-] TABLE ACCESS STORAGE FULL |          | 31    | 126K   | 3,787 |           | 503 | 8,586K | 1 |

一時表領域のIOをしている場合、  
まず最初に確認するのは  
予測した行数と  
実際行の乖離

予測した行数: 12.6万行  
実際行: 858.6万行

予測した行数 < 実際行の  
場合(小さく  
見積もっていた場合)、  
PX SEND BROADCAST  
していないか確認

# パラレル実行時の分散方法の違い

## PX SEND BROADCAST

- 小さい表と大きい表を結合するときに使う
- 大きい表を読み取るスレーブ・プロセス全員に同じ小さい表を送る
- 小さい表のサイズ × パラレル度の一時領域が必要

## PX SEND HASH

- 大きい表と大きい表を結合するときに使う
- 2つの表をそれぞれハッシュ分割させて結合する
- 大きい表のサイズ分の一時領域が必要

# 結論: 表Cの統計情報の精度が悪い

## 対策

- ヒストグラムがない場合はヒストグラムを作成
- PQ\_DISTRIBUTE(  
    c, HASH, HASH  
) ヒント
- 拡張統計がない場合は拡張統計を作成
  - 自動列グループ検出を使うと簡単に作成可能
    - DBMS\_STATS.SEED\_COL\_USAGE()
    - DBMS\_STATS.REPORT\_COL\_USAGE()
    - DBMS\_STATS.CREATE\_EXTENDED\_STATS('ownname', 'tabname')
      - REPORT\_COL\_USAGEしておく、3個目の引数が不要に

# まとめ

- Oracle Database はAWRレポートやリアルタイムSQL監視レポートだけでチューニングできる
- Oracle Database はトレースなどのオーバーヘッドの大きい分析を再度、別環境で行なう必要がない
- Oracle Database はチューニングしやすいところが大きな強み

# Integrated Cloud

## Applications & Platform Services

ORACLE®