Parallel Execution Plans

Jonathan Lewis jonathanlewis.wordpress.com www.jlcomp.demon.co.uk

My History

Independent Consultant

33+ years in IT28+ using Oracle (5.1a on MSDOS 3.3

Strategy, Design, Review, Briefings, Educational, Trouble-shooting



Oracle author of the year 2006 Select Editor's choice 2007 UKOUG Inspiring Presenter 2011 ODTUG 2012 Best Presenter (d/b) UKOUG Inspiring Presenter 2012 UKOUG Lifetime Award (IPA) 2013 Member of the Oak Table Network Oracle ACE Director

<section-header>

O1 visa for USA

Jonathan Lewis
© 2011 - 2016Many slides have a foot-note. This is just two lines summarizing the highlights of the
slide so that you have a reference when reading the handouts at a later date.

PX Plans p. 2 / 34

Topics

- What are we looking for in a plan
 - Order of operation (row source generation)
 - Resource usage
 - Early elimination of data
- What tools can we use
 - dbms_xplan
 - v\$pq_tqstat
 - Extended sql_trace, or equivalent
 - v\$sql_monitor if licensed (diagnostic + performance)

Terminology

- **QC:** "Query coordinator" the process controlling the query (and passing data to the front end)
- **PX Server:** single process used in parallel query a.k.a Parallel server, Parallel Query Slave, PQ slave, PX slave
- Slave Set: A set of PX Servers performing one operation of an execution plan commonly a single query will use two sets of PX servers
- **DOP:** "degree of parallelism" number of slaves in *each* slave set involved in a full parallel execution plan
- **Table Queue:**Logical communication channel between two sets of
slaves, or from a slave set to the QC

a.k.a Virtual table

- **DFO:** "data flow operation" the set of actions that moves data through a single table queue
- **DFO tree:** Set of DFOs moving data from its source to the QC

Big Problem

SMALL PRINT

Jonathan	Lewis
© 2011 -	2016

Sample Data (a)

create table **t1** as

select

© 2011 - 2016

rownum	id,
to_char(rownum)	small_vc,
rpad('x',100)	padding

- from all_objects
- where *rownum* <= 70;

alter table t1 add constraint t1 pk primary key(id);

```
begin
   dbms_stats.gather_table_stats(
        user,
        't1',
        method_opt => 'for all columns size 1'
   );
   end;
Jonathan Lewis
```

Repeat for matching t2 and t3

PX	Plans
p.	6/34

Sample Data (b)

© 2011 - 2016

create	e table t4 as	
select	;	
	t1.id	id1,
	t2.id	id2,
	t3.id	id3,
	<pre>rpad(rownum,10)</pre>	small_vc,
	rpad('x',100)	padding
from		
	t1, t2, t3	343,000 rows
;		
begin		
dbm	s_stats.gather_table_s	stats(
	user,	
	't4',	
	<pre>method_opt => 'for a</pre>	ll columns size 1'
);		
end;		
Jonathan Lewis		

PX Plans p. 7 / 34

Sample Query (serial)

```
select
            /*+ gather plan statistics */
            count(t1.small vc), count(t2.small vc),
            count(t3.small vc), count(t4.small vc)
    from
            t4,
            t1,
            t2,
            t3
    where
            t1.id = t4.id1
            t2.id = t4.id2
    and
           t3.id = t4.id3
    and
                                                       type mismatch !!!
            t1.small vc in (1, 2, 3)
    and
    and
            t2.small vc in (1,2,3,4)
    and
            t3.small vc in (1,2,3,4,5)
    ;
Jonathan Lewis
© 2011 - 2016
```

Sample Query (serial plan)

select * from table(dbms_xplan.display_cursor(null,null,'allstats last'));

	Id	Operation		Name		Starts	E-Rows		A-Rows	
I	0	SELECT STATEMENT				1			1	
Ι	1	SORT AGGREGATE				1	1		1	
*	2	HASH JOIN				1	56		60	
*	- 3	TABLE ACCESS FULL		ΤЗ		1	5		5	
*	4	HASH JOIN				1	810		840	
*	5	TABLE ACCESS FULL		Т2		1	4		4	
*	6	HASH JOIN				1	14491		14700	
*	7	TABLE ACCESS FULI	_	Τ1		1	3		3	
	8	TABLE ACCESS FULI	_	Т4		1	343¥	ζ	343K	

```
leading(t4 t1 t2 t3)
use_hash(t1) swap_join_inputs(t1)
use_hash(t2) swap_join_inputs(t2)
use_hash(t3) swap_join_inputs(t3)
```

Jonathan Lewis
© 2011 - 2016We can read the plan by "first child - recursive descent". The order of action is: scan
and hash t3, scan and hash t2, scan and hash t1, scan t4 and probe x3

PX Plans p. 9 / 34

Sample Query (serial trace)

alter system flush buffer_cache; alter session set events '10046 trace name context forever, level 8';

Tablescan table t3 WAIT #: nam='db file sequential read' ela= 2207 f#=7 b#=640 bs=1 obj#=**235626** WAIT #: nam='db file scattered read' ela= 570 f#=7 b#=641 bs=2 obj#=235626

 Tablescan table t2

 WAIT #: nam='db file sequential read' ela=
 458 f#=7 b#=384 bs=1 obj#=235624

 WAIT #: nam='db file scattered read' ela=
 387 f#=7 b#=385 bs=2 obj#=235624

Tablescan table t1 WAIT #: nam='db file sequential read' ela= 524 f#=7 b#=128 bs=1 obj#=**235622** WAIT #: nam='db file scattered read' ela= 477 f#=7 b#=129 bs=2 obj#=235622

 Tablescan table t4 (direct)

 WAIT #: nam='db file sequential read' ela= 502 f#=7 b#=896 bs= 1 obj#=235628

 WAIT #: nam='direct path read'
 ela= 1658 f= 7 fd=897 bc=127 obj#=235628

WAIT #140096457765816: nam='direct path read'

Jonathan Lewis
© 2011 - 2016As a little check for order of operation, the 10046 trace file (flushing the buffer cache
before doing the test) can show us the physical read waits.PX Plans
p. 10 / 34

Going Parallel (hash/hash)

```
select
       /*+
               gather plan statistics
               leading(t4 t1 t2 t3)
               parallel(t4,2) full(t4) parallel(t1,2) full(t1)
               parallel(t2,2) full(t2) parallel(t3,2) full(t3)
               use hash(t1) swap join inputs(t1)
               pq distribute (t1 hash hash)
               use hash(t2) swap join inputs(t2)
               pq distribute(t2 hash hash)
               use hash(t3) swap join inputs(t3)
               pq distribute(t3 hash hash)
       */
       count(t1.small vc), count(t2.small vc),
       count(t3.small vc), count(t4.small vc)
from
       ...
```

Jonathan Lewis
© 2011 - 2016parallel/full - force a parallel full tablescan. use_hash/swap_join_inputs - do a hash
join with swap; when tN is the "next" table in the join *hash distribute* both inputs

Going Parallel (broadcast)

© 2011 - 2016

```
select
            /*+
                    gather plan statistics
                    leading(t4 t1 t2 t3)
                    parallel(t4,2) full(t4) parallel(t1,2) full(t1)
                    parallel(t2,2) full(t2) parallel(t3,2) full(t3)
                    use hash(t1) swap join inputs(t1)
                    pq distribute (t1 none broadcast)
                    use hash(t2) swap join inputs(t2)
                    pq distribute(t2 none broadcast)
                    use hash(t3) swap join inputs(t3)
                    pq distribute(t3 none broadcast)
            */
            count(t1.small vc), count(t2.small vc),
            count(t3.small vc), count(t4.small vc)
    from
            ...
           parallel/full - force a parallel full tablescan. use hash/swap join inputs - do a hash
Jonathan Lewis
```

join with swap; when *tN* is the next table in the join *broadcast* it

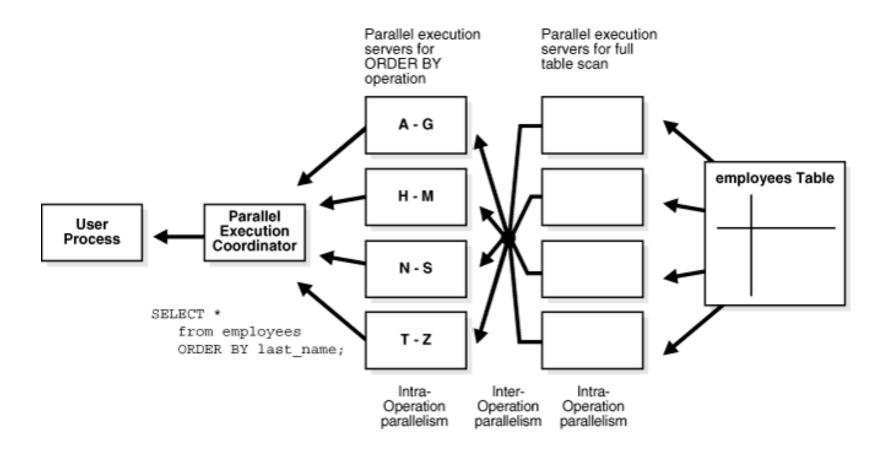
PX Plans p. 12 / 34

<u> </u>	Id		Operation	Name	Rows	ΤQ	IN-OUT	PQ Distrib
	0		SELECT STATEMENT					
	1		SORT AGGREGATE		1			
	2	T	PX COORDINATOR					
	3		PX SEND QC (RANDOM)	:TQ10003	1	Q1,03	P->S	QC (RAND)
	4		SORT AGGREGATE		1	Q1,03	PCWP	
I	5	I	HASH JOIN		56	Q1,03	PCWP	I
	6		PX RECEIVE		5	Q1,03	PCWP	
	7	I	PX SEND BROADCAST	:TQ10000	5	Q1,00	P->P	BROADCAST
	8		PX BLOCK ITERATOR		5	Q1,00	PCWC	
I	9	Ι	TABLE ACCESS FULL	т3	5	Q1,00	PCWP	I
I	10	Ι	HASH JOIN	l	810	Q1,03	PCWP	I
	11	I	PX RECEIVE		4	Q1,03	PCWP	
	12	I	PX SEND BROADCAST	:TQ10001	4	Q1,01	P->P	BROADCAST
	13	I	PX BLOCK ITERATOR	l	4	Q1,01	PCWC	
I	14	Ι	TABLE ACCESS FULL	т2	4	Q1,01	PCWP	I
I	15	Ι	HASH JOIN	I	14491	Q1,03	PCWP	I
	16		PX RECEIVE	I	3	Q1,03	PCWP	
	17		PX SEND BROADCAST	:TQ10002	3	Q1,02	P->P	BROADCAST
	18		PX BLOCK ITERATOR	I	3	Q1,02	PCWC	
I	19	Ι	TABLE ACCESS FULL	T1	3	Q1,02	PCWP	I
	20	I	PX BLOCK ITERATOR	I	343K	Q1,03	PCWC	I
<u> </u>	21	I	TABLE ACCESS FULL	т4	343K	Q1,03	PCWP	<u> </u>

Jonathan Lewis © 2011 - 2016 We now have 22 lines instead of 9 but, between all the send/receive operations we can still see the shape of the four table hash join with the original join order.

PX Plans p. 13 / 34

Parallel Images

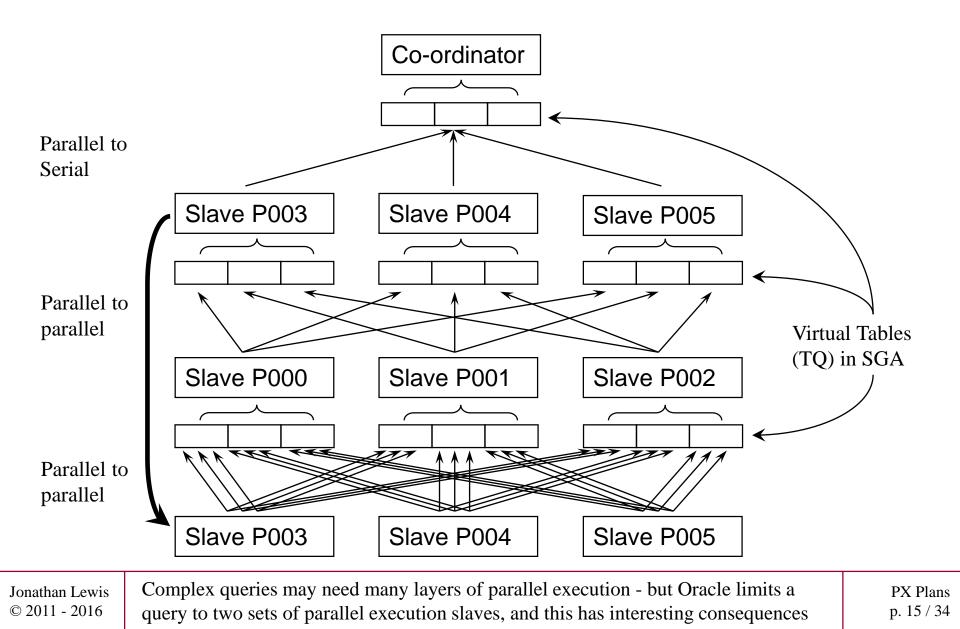


Oracle® Database VLDB and Partitioning Guide Ch. 8

Jonathan LewisConveniently this simple example shows just two send/receive pairs: from slave set 1© 2011 - 2016to slave set 2, then from slave set to the query coordinator. Real-life is more complex

PX Plans p. 14 / 34

Parallel Execution - visual



	Id		Operation	Name	Rows	ΤQ	IN-OUT	PQ Distrib
	0		SELECT STATEMENT					
	1		SORT AGGREGATE		1			
	2		PX COORDINATOR					
	3		PX SEND QC (RANDOM)	:TQ10003	1	Q1,03	P->S	QC (RAND)
	4		SORT AGGREGATE		1	Q1,03	PCWP	
	5		HASH JOIN		56	Q1,03	PCWP	
	6		PX RECEIVE		5	Q1,03	PCWP	
I	7	Ι	PX SEND BROADCAST	:TQ10000	5	Q1,00	<i>P->P</i>	BROADCAST
I	8	Ι	PX BLOCK ITERATOR		5	Q1,00	PCWC	l I
I	9	I	TABLE ACCESS FULL	ТЗ	5	Q1,00	PCWP	l I
	10		HASH JOIN		810	Q1,03	PCWP	
	11		PX RECEIVE		4	Q1,03	PCWP	
I	12	I	PX SEND BROADCAST	:TQ10001	4	Q1,01	<i>P->P</i>	BROADCAST
I	13	I	PX BLOCK ITERATOR		4	Q1,01	PCWC	l I
I	14	I	TABLE ACCESS FULL	T2	4	Q1,01	PCWP	l I
	15		HASH JOIN		14491	Q1,03	PCWP	
	16		PX RECEIVE		3	Q1,03	PCWP	
I	17	I	PX SEND BROADCAST	:TQ10002	3	Q1,02	<i>P->P</i>	BROADCAST
I	18	I	PX BLOCK ITERATOR		3	Q1,02	PCWC	I.
I	19	I	TABLE ACCESS FULL	T1	3	Q1,02	PCWP	I.
	20		PX BLOCK ITERATOR		343K	Q1,03	PCWC	
	21		TABLE ACCESS FULL	Τ4	343K	Q1,03	PCWP	<u> </u>

Jonathan Lewis © 2011 - 2016

For parallel queries we have to follow the "virtual tables", known as *"table queues"*. The order of operation follows the sequence of generating TQs (Name column.)

PX Plans p. 16 / 34

	Id Operation	Name	Rows	TQ IN-OUT PQ Distrib
--	----------------	------	------	------------------------

	6		PX RECEIVE				5	Ι	Q1,03 PCWH	,		
	7	Ι	PX SEND BROADCAST	I	:TQ10000	1	5	Ι	Q1,00 P->1	'	BROADCAST	Ι
	8	Ι	PX BLOCK ITERATOR	I		1	5	Ι	Q1,00 PCW0	:		Ι
	9	Ι	TABLE ACCESS FULL	I	тЗ	I	5	Ι	Q1,00 PCW1	'		Ι
	11		PX RECEIVE				4		Q1,03 PCWH	'		
	12	Ι	PX SEND BROADCAST	I	:TQ10001	1	4	Ι	Q1,01 P->1	'	BROADCAST	Ι
	13	I	PX BLOCK ITERATOR	I		I	4	Ι	Q1,01 PCW0	:		Ι
I	14	Ι	TABLE ACCESS FULL	I	Т2	I	4	Ι	Q1,01 PCW1	'		Ι
	16		PX RECEIVE				3	Ι	Q1,03 PCWH	'		
	17	Ι	PX SEND BROADCAST	I	:TQ10002	I	3	Ι	Q1,02 P->1	'	BROADCAST	Ι
	18	Ι	PX BLOCK ITERATOR	I		I	3	Ι	Q1,02 PCW0	:		Ι
	19	Ι	TABLE ACCESS FULL	I	T1	I	3	Ι	Q1,02 PCW1	'		I

Jonathan Lewis
© 2011 - 2016In this case the order of operation matches the serial plan. All three tables are inputs
to the same (TQ10003) virtual table - so a single slave set must be receiving them.

	Id		Operation		Name		Rows	TQ]	IN-OUT	PQ	Distrib
	0		SELECT STATEMENT									
	1		SORT AGGREGATE				1					
	2		PX COORDINATOR									1
Ι	3	Ι	PX SEND QC (RANDOM)	Ι	:TQ10003	I	1	Q1,03	Ι	P->S	QC	(RAND)
Ι	4	Ι	SORT AGGREGATE	Ι		I	1	Q1,03	Ι	PCWP		I
Ι	5	Ι	HASH JOIN	Ι		I	56	Q1,03	Ι	PCWP		I
Ι	6	Ι	PX RECEIVE	Ι		I	5	Q1,03	Ι	PCWP		I
 	10 11		HASH JOIN PX RECEIVE	 			810 4		-	PCWP PCWP		
 	15 16		HASH JOIN PX RECEIVE				14491 3		-	PCWP PCWP		
•	20 21	 	PX BLOCK ITERATOR TABLE ACCESS FULL	 	T4		343K 343K		•	PCWC PCWP		
	Jonathan LewisThink of the <i>name</i> column as the virtual table being used as a pipeline, then the TQ © 2011 - 2016column tells you how a set of slaves finds data for that virtual table.											PX Plans p. 18 / 34

PQ TQ stats (v\$pq_tqstat)

select

	dfo_number,		
	tq_id,		
	server_type,		producer/consumer/ranger
	instance,		for RAC
	process,		PNNN
	num_rows		
from			
	v\$pq_tqstat		
order b	ру		
	dfo_number,		
	tq_id,		
	server_type de	sc,	
	instance,		
	process		
•			

;

Jonathan Lewis TQ © 2011 - 2016 the

TQ10003 in the plan can be aligned with dfo_number = 1, $tq_id = 3$ (We will ignore the significance of DFO's (Data Flow Operation trees) at present.

PX Plans p. 19 / 34

PQ Stats (broadcast)

DFO	NUMBER	ΤQ	ID	SERVER	TYPE	INSTANC	E	PROCESS	NUM	ROWS		
	1		0	Produce	r (s e	end)	1	P002		10		
							1	P003		0	t3	scan
				Consume	r (r e	eceive)	1	P000		5		
							1	P001		5	t3	hash
			_	_			_					
			1	Produce	r		1	P002		8		
							1	P003		0	t2	scan
				Consume	r		1	P000		4		
							1	P001		4	t2	hash
			2	Produce	r		1	P002		6		
			2	IIOddee	-			P003		0	t1	scan
				Consume	r			P000		3	-	
							1	P001		3	t1	hash
			-				_					
			3	Produce	r		1	P000		1	t4	scan,
							1	P001		1	pr	obe & ct
				Consume	r		1	QC		2		

group by and order by result in rows where the QC operates as server_type = Ranger

Jonathan Lewis	Running parallel 2: we see three consecutive jobs as p002 & p003 broadcast N x 2	PX Plans
© 2011 - 2016	rows to p000 & p001; then p000 & p001 produce the result of all three joins.	p. 20 / 34

Trace files (broadcast)

In the previous slide slaves p000 and p001 scanned table t4 - so what do their trace files say about the work done - the estimate was to generate 343,000 rows before joining

	20	PX BLOCK ITERATOR		I	343K	Q1,03 PCWC
	21	TABLE ACCESS FULL	T4	I	343K	Q1,03 PCWP

P000

STAT #N id=20 *cnt=40* pid=15 pos=2 obj=0 op='PX BLOCK ITERATOR(card=343000)' STAT #N id=21 *cnt=40* pid=20 pos=1 obj=235635 op='TABLE ACCESS FULL T4(card=343000)'

P001

STAT #N id=20 *cnt=20* pid=15 pos=2 obj=0 op='PX BLOCK ITERATOR(card=343000)' STAT #N id=21 *cnt=20* pid=20 pos=1 obj=235635 op='TABLE ACCESS FULL T4(card=343000)'

This shows a total of 60 rows returned from the table scan of t4 *before* the first join. This is the effect of *Bloom* filtering.

On the Exadata database machine the Bloom filters can be sent to the storage server The storage server can use storage indexes and smart scans to minimise disk and network load

Jonathan Lewis
© 2011 - 2016A Bloom filter changes a join into a predicate. It eliminates (most of the) data that
you don't want, allows all the data you do want - but may return some unwanted data

Parallel display_cursor()

0 SELECT STATEMENT	1 1 2 0 2 60
2 PX COORDINATOR	2
3 PX SEND QC (RANDOM) :TQ10003 1 Q1,03 P->S QC (RAND) 4 SORT AGGREGATE 1 Q1,03 PCWP * 5 HASH JOIN 56 Q1,03 PCWP	2
4 SORT AGGREGATE 1 Q1,03 PCWP * 5 HASH JOIN 56 Q1,03 PCWP	2
* 5 HASH JOIN 56 Q1,03 PCWP	
	601
6 PX RECEIVE 5 01,03 PCWP	001
	10
7 PX SEND BROADCAST :TQ10000 5 Q1,00 P->P BROADCAST	0
8 PX BLOCK ITERATOR 5 Q1,00 PCWC	5
* 9 TABLE ACCESS FULL T3 5 Q1,00 PCWP	5
*10 HASH JOIN 810 Q1,03 PCWP	60
11 PX RECEIVE 4 Q1,03 PCWP	8
12 PX SEND BROADCAST :TQ10001 4 Q1,01 P->P BROADCAST	0
13 PX BLOCK ITERATOR 4 Q1,01 PCWC	4
*14 TABLE ACCESS FULL T2 4 Q1,01 PCWP	4
*15 HASH JOIN 14491 Q1,03 PCWP	60
16 PX RECEIVE 3 Q1,03 PCWP	6
17 PX SEND BROADCAST :TQ10002 3 Q1,02 P->P BROADCAST	0
18 PX BLOCK ITERATOR 3 Q1,02 PCWC	3
*19 TABLE ACCESS FULL T1 3 Q1,02 PCWP	3
20 PX BLOCK ITERATOR 343K Q1,03 PCWC	60
*21 TABLE ACCESS FULL T4 343K Q1,03 PCWP	601

Jonathan Lewis © 2011 - 2016

After running the query just once I've used the format option *'allstats parallel'* without *''last''* to get this output (and then I've deleted several columns).

PX Plans p. 22 / 34

Predicate Information (identified by operation id):

- 5 access("T3"."ID"="T4"."ID3")
- 9 access(:Z>=:Z AND :Z<=:Z) -- check rowid ranges
 - filter((TO_NUMBER("T3"."SMALL_VC")=1 OR TO_NUMBER("T3"."SMALL_VC")=2
 OR TO_NUMBER("T3"."SMALL_VC")=3 OR TO_NUMBER("T3"."SMALL_VC")=4
 OR TO NUMBER("T3"."SMALL_VC")=5))
- 10 access("T2"."ID"="T4"."ID2")
- 14 access(:Z > =: Z AND :Z < =: Z)
- 19 access(:Z > : Z AND : Z < : Z)
 - filter((TO_NUMBER("T1"."SMALL_VC")=1 OR TO_NUMBER("T1"."SMALL_VC")=2 OR TO NUMBER("T1"."SMALL VC")=3))
- 21 access(:Z>=:Z AND :Z<=:Z)

filter(SYS_OP_BLOOM_FILTER_LIST(

SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID1"),

SYS OP BLOOM FILTER(:BF0000, "T4". "ID2"),

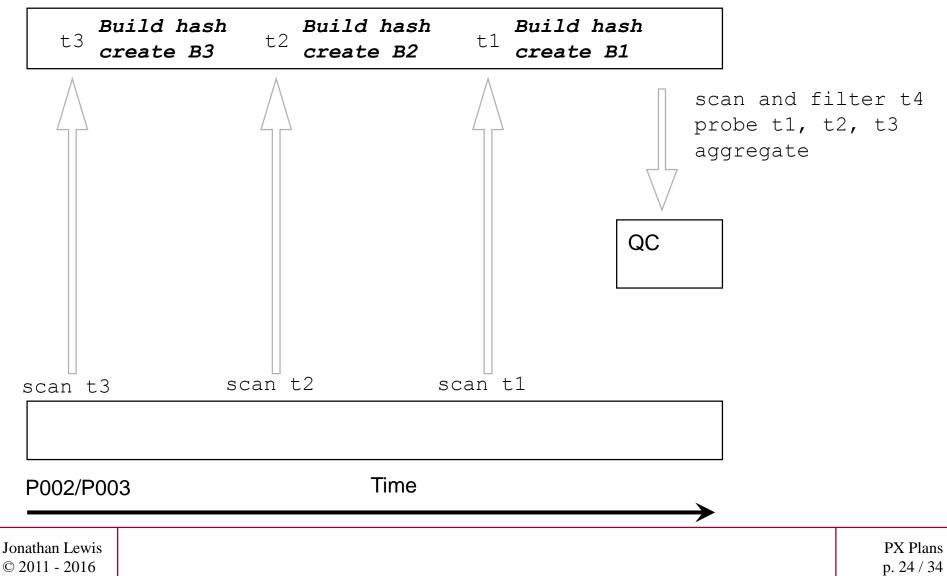
SYS OP BLOOM FILTER(:BF0000, "T4". "ID3")

)) -- Blooom filters from all three dimensions used during tablescan

Jonathan Lewis	In 11g the predicate section shows that three Bloom filters were used during the <i>t4</i>
© 2011 - 2016	tablescan. They identify which column is filtered on, but the BF numbering is odd.

Graphic (broadcast)

P000/P001



12c Broadcast plan

Id	Operation	Name	Rows	Bytes	Cost	TQ	IN-OUT	PQ Distr	ib
0	SELECT STATEMENT				351				
1	SORT AGGREGATE		1	38					
2	PX COORDINATOR								
3	PX SEND QC (RANDOM)	:TQ10000	1	38		Q1,00	P->S	QC (RAND)
4	SORT AGGREGATE		1	38		Q1,00	PCWP		
* 5	HASH JOIN		56	2128	351	Q1,00	PCWP		
6	JOIN FILTER CREATE	:BF0000	5	30	2	Q1,00	PCWP		
* 7	TABLE ACCESS FULL	Т3	5	30	2	Q1,00	PCWP		
* 8	HASH JOIN		810	25920	349	Q1,00	PCWP		
9	JOIN FILTER CREATE	:BF0001	4	24	2	Q1,00	PCWP		
*10	TABLE ACCESS FULL	Т2	4	24	2	Q1,00	PCWP		
*11	HASH JOIN		14491	367K	347	Q1,00	PCWP		
12	JOIN FILTER CREATE	:BF0002	3	18	2	Q1,00	PCWP		I
*13	TABLE ACCESS FULL	Т1	3	18	2	Q1,00	PCWP		
14	JOIN FILTER USE	:BF0000	343K	6699K	345	Q1,00	PCWP		I
15	JOIN FILTER USE	:BF0001	343K	6699K	345	Q1,00	PCWP		I
16	JOIN FILTER USE	:BF0002	343K	6699K	345	Q1,00	PCWP		
17	PX BLOCK ITERATOR		343K	6699K	345	Q1,00	PCWC		
*18	TABLE ACCESS FULL	Т4	343K	6699K	345	Q1,00	PCWP		

Jonathan Lewis
© 2011 - 2016There are many differences in the 12c, which changes dramatically, caching scans,
avoiding many table queues, and showing us where the filters are created and used.

PX Plans p. 25 / 34

Id	Operation	Name	Rows	Time	ТQ	IN-OUT	PQ Distrib
0	SELECT STATEMENT						
1	SORT AGGREGATE		1		l	I I	1
2	PX COORDINATOR		I I		l	I I	1
3	PX SEND QC (RANDOM)	:TQ10006	1		Q1,06	P->S	QC (RAND)
4	SORT AGGREGATE		1		Q1,06	PCWP	I
* 5	HASH JOIN		56	00:00:29	Q1,06	PCWP	I
6	JOIN FILTER CREATE	:BF0000	5	00:00:01	Q1,06	PCWP	I
7	PX RECEIVE		5	00:00:01	Q1,06	PCWP	I
8	PX SEND HASH	:TQ10004	5	00:00:01	Q1,04	P->P	HASH
9	PX BLOCK ITERATOR		5	00:00:01	Q1,04	PCWC	I
* 10	TABLE ACCESS FULL	<u>T3</u>	5	00:00:01	Q1,04	PCWP	I
11	PX RECEIVE		810	00:00:29	Q1,06	PCWP	I
12	PX SEND HASH	:TQ10005	810	00:00:29	Q1,05	P->P	HASH
13	JOIN FILTER USE	:BF0000	810	00:00:29	Q1,05	PCWP	I
* 14	HASH JOIN BUFFERED		810	00:00:29	Q1,05	PCWP	I
15	JOIN FILTER CREATE	:BF0001	4	00:00:01	Q1,05	PCWP	I
16	PX RECEIVE		4	00:00:01	Q1,05	PCWP	I
17	PX SEND HASH	:TQ10002	4	00:00:01	Q1,02	P->P	HASH
18	PX BLOCK ITERATOR		4	00:00:01	Q1,02	PCWC	I
* 19	TABLE ACCESS FULL	<u>T2</u>	4	00:00:01	Q1,02	PCWP	I
20	PX RECEIVE		14491	00:00:29	Q1,05	PCWP	I
21	PX SEND HASH	:TQ10003	14491	00:00:29	Q1,03	P->P	HASH
22	JOIN FILTER USE	:BF0001	14491	00:00:29	Q1,03	PCWP	I
* 23	HASH JOIN BUFFERED		14491	00:00:29	Q1,03	PCWP	I
24	JOIN FILTER CREATE	:BF0002	3	00:00:01	Q1,03	PCWP	I
25	PX RECEIVE		3	00:00:01	Q1,03	PCWP	I
26	PX SEND HASH	:TQ10000	3	00:00:01	Q1,00	P->P	HASH
27	PX BLOCK ITERATOR		3	00:00:01	Q1,00	PCWC	I
* 28	TABLE ACCESS FULL	<u>T1</u>	3	00:00:01	Q1,00	PCWP	I
29	PX RECEIVE		343K	00:00:29	Q1,03	PCWP	I
30	PX SEND HASH	:TQ10001	343K	00:00:29	Q1,01	P->P	HASH
31	JOIN FILTER USE	:BF0002	343K	00:00:29	Q1,01	PCWP	I
32	PX BLOCK ITERATOR		343K	00:00:29	Q1,01	PCWC	I
* 33	TABLE ACCESS FULL	т4	343K	00:00:29	Q1,01	PCWP	<u> </u>

Jonathan Lewis © 2011 - 2016 We've now gone from 22 lines to 34 lines, but we can still see the shape and order of the original four table hash join. (*Hash Join Buffered* is a threat!)

PX Plans p. 26 / 34

Id	Operation	Name	Rows	Time	I TQ	IN-OUT	PQ Distrib	<u> </u>	
0	SELECT STATEMENT				1				
1	SORT AGGREGATE		1		I				
2	PX COORDINATOR				I				
3	PX SEND QC (RANDOM)	<u>: TQ10006</u>	1		Q1,06	P->S	QC (RAND)		
4	SORT AGGREGATE		1		Q1,06	PCWP			
* 5	HASH JOIN		56	00:00:29	Q1,06	PCWP			
6	JOIN FILTER CREATE	:BF0000	5	00:00:01	Q1,06	PCWP			
7	PX RECEIVE		5	00:00:01	Q1,06	PCWP			
8	PX SEND HASH	<u>: TQ10004</u>	5	00:00:01	Q1,04	P->P	HASH		4
9	PX BLOCK ITERATOR		5	00:00:01	Q1,04	PCWC			
* 10	TABLE ACCESS FULL	Т3	5	00:00:01	Q1,04	PCWP			
11	PX RECEIVE		810	00:00:29	Q1,06	PCWP			
12	PX SEND HASH	<u>: TQ10005</u>	810	00:00:29	Q1,05	P->P	HASH		/
13	JOIN FILTER USE	:BF0000	810	00:00:29	Q1,05	PCWP			
* 14	HASH JOIN BUFFERED		810	00:00:29	Q1,05	PCWP			
15	JOIN FILTER CREATE	:BF0001	4	00:00:01	Q1,05	PCWP			
16	PX RECEIVE		4	00:00:01	Q1,05	PCWP			
17	PX SEND HASH	: TQ10002	4	00:00:01	Q1,02	P->P	HASH	∖ ∖	
18	PX BLOCK ITERATOR		4	00:00:01	Q1,02	PCWC			
* 19	TABLE ACCESS FULL	Т2	4	00:00:01	Q1,02	PCWP			
20	PX RECEIVE		14491	00:00:29	Q1,05	PCWP			
21	PX SEND HASH	<u>: TQ10003</u>	14491	00:00:29	Q1,03	P->P	HASH	⊢)
22	JOIN FILTER USE	:BF0001	14491	00:00:29	Q1,03	PCWP		1	
* 23	HASH JOIN BUFFERED		14491	00:00:29	Q1,03	PCWP		1	
24	JOIN FILTER CREATE	:BF0002	3	00:00:01	Q1,03	PCWP		1	
25	PX RECEIVE		3	00:00:01	Q1,03	PCWP		L	
26	PX SEND HASH	<u>: TQ10000</u>	3	00:00:01	Q1,00	P->P	HASH		
27	PX BLOCK ITERATOR		3	00:00:01	Q1,00	PCWC			
* 28	TABLE ACCESS FULL	Т1	3	00:00:01	Q1,00	PCWP			
29	PX RECEIVE		343K	00:00:29	Q1,03	PCWP		I	
30	PX SEND HASH	<u>: TQ10001</u>	343K	00:00:29	Q1,01	P->P	HASH	▶	
31	JOIN FILTER USE	:BF0002	343K	00:00:29	Q1,01	PCWP			
32	PX BLOCK ITERATOR		343K	00:00:29	Q1,01	PCWC			
* 33	TABLE ACCESS FULL	Т4	343K	00:00:29	Q1,01	PCWP		<u> </u>	

Jonathan Lewis © 2011 - 2016 We now have *seven* table queues to follow. Notice how they don't follow a simple consecutive pattern up the plan, though.

Id	Operation	Name	Rows	Time	I TQ	IN-OUT	PQ Distrib	
0	SELECT STATEMENT							_
1	SORT AGGREGATE		1					
2	PX COORDINATOR							
3	PX SEND QC (RANDOM)	:TQ10006	1		Q1,06	P->S	QC (RAND)	
4	SORT AGGREGATE		1		Q1,06	PCWP		
* 5	HASH JOIN		56	00:00:29	Q1,06	PCWP		
6	JOIN FILTER CREATE	:BF0000	5	00:00:01	Q1,06	PCWP		
7	PX RECEIVE		5	00:00:01	Q1,06	PCWP		
8	PX SEND HASH	:TQ10004	5	00:00:01	Q1,04	P->P	HASH	1
9	PX BLOCK ITERATOR		5	00:00:01	Q1,04	PCWC		
* 10	TABLE ACCESS FULL	ТЗ	5	00:00:01	Q1,04	PCWP		
11	PX RECEIVE		810	00:00:29	Q1,06	PCWP		
12	PX SEND HASH	:TQ10005	810	00:00:29	Q1,05	P->P	HASH	I
13	JOIN FILTER USE	:BF0000	810	00:00:29	Q1,05	PCWP		
* 14	HASH JOIN BUFFERED		810	00:00:29	Q1,05	PCWP		
15	JOIN FILTER CREATE	:BF0001	4	00:00:01	Q1,05	PCWP		
16	PX RECEIVE		4	00:00:01	Q1,05	PCWP		1
17	PX SEND HASH	:TQ10002	4	00:00:01	Q1,02	P->P	HASH	1
18	PX BLOCK ITERATOR		4	00:00:01	Q1,02	PCWC		1
* 19	TABLE ACCESS FULL	T2	4	00:00:01	Q1,02	PCWP		1
20	PX RECEIVE		14491	00:00:29	Q1,05	PCWP		We create the filter
21	PX SEND HASH	:TQ10003	14491	00:00:29	Q1,03	P->P	HASH	
22	JOIN FILTER USE	:BF0001	14491	00:00:29	Q1,03	PCWP		<i>after</i> we receive
* 23	HASH JOIN BUFFERED		14491	00:00:29	Q1,03	PCWP		the build table (T1)
24	JOIN FILTER CREATE	:BF0002	3	00:00:01	Q1,03	PCWP		
25	PX RECEIVE		3	00:00:01	Q1,03	PCWP		
26	PX SEND HASH	:TQ10000	3	00:00:01	Q1,00	P->P	HASH	1
27	PX BLOCK ITERATOR		3	00:00:01	Q1,00	PCWC		1
* 28	TABLE ACCESS FULL	T1	3	00:00:01	Q1,00	PCWP		I
29	PX RECEIVE		343K	00:00:29	Q1,03	PCWP		I
30	PX SEND HASH	:TQ10001	343K	00:00:29	Q1,01	P->P	HASH	
31	JOIN FILTER USE	:BF0002	343K	00:00:29	Q1,01	PCWP		We use it <i>before</i> we
32	PX BLOCK ITERATOR	l	343K	00:00:29	Q1,01	PCWC		send the probe table
* 33	TABLE ACCESS FULL	T4	343K	00:00:29	Q1,01	PCWP		

Jonathan Lewis © 2011 - 2016 The plan includes several pairs of lines showing the creation and use of Bloom filters (We have to ignore the BF numbers as they don't agree with the order of creation).

PX Plans p. 28 / 34

Predicate Information (identified by operation id):

- 5 access("T3"."ID"="T4"."ID3")
- 10 access(:Z>=:Z AND :Z<=:Z)
 - filter((TO_NUMBER("T3"."SMALL_VC")=1 OR TO_NUMBER("T3"."SMALL_VC")=2 OR TO_NUMBER("T3"."SMALL_VC")=3))
- 14 access("T2"."ID"="T4"."ID2")
- 19 access(:Z>=:Z AND :Z<=:Z)
 - filter((TO_NUMBER("T2"."SMALL_VC")=1 OR TO_NUMBER("T2"."SMALL_VC")=2 OR TO NUMBER("T2"."SMALL VC")=3))
- 23 access("T1"."ID"="T4"."ID1")
- 28 access(:Z>=:Z AND :Z<=:Z)
 - filter((TO_NUMBER("T1"."SMALL_VC")=1 OR TO_NUMBER("T1"."SMALL_VC")=2 OR TO NUMBER("T1"."SMALL VC")=3))

33 - access(:Z>=:Z AND :Z<=:Z)

filter(SYS_OP_BLOOM_FILTER(:BF0000,"T4"."ID1"))

Jonathan Lewis
© 2011 - 2016Although the plan says we created and used three Bloom filters the predicate section
only reports using one of them. We need to check execution stats.

OEM monitor (11g)

	i 🔒 https://linux11:1158/em/console/database/ir	Instance/sqlMonitorDetail?target=t	est.localdomain&type=oracl	.e_database&sqlId=dg5nvnf1dws	s34&start=2016.5.30.19.3! 🗸 🧭 📿 Search	☆ 🖻	e 🛡 🕇 🏫	1 9
Number Note	k Web App 🛞 Dashboard k Oracle S 🍠 Twitter 👔	🗧 All Things Oracle - Co 🧧 (OTN DB General 🖸 Space	: PL/SQL Oracl My Oracle	e Support FL oracle-I Mailing List Arc G Google			
Number Note								
n Nome Estimate/leve Oriel Terms(0):100:000 Nome (No. Nome (No. Nome (No. Oriel Active (No. Nome (No. Oriel Active (No. Nome (No. Oriel Active (No. Nome (No.	atistics 🖓 Parallel 📐 Activity							
Act #Arthon I <t< td=""><td>Value 3607894828</td><td></td><td></td><td></td><td></td><td>𝗭 TIP: Right mouse click on the table ₽'</td><td>allows to toggle between IO Rer</td><td>quests and IO.</td></t<>	Value 3607894828					𝗭 TIP: Right mouse click on the table ₽'	allows to toggle between IO Rer	quests and IO.
orr Addetative results of Image of Ima		Name	Estimated Rows	Cost Timeline(0.120695s)		CPU Activity %	Wait Activity %	
Operation Set and a construction Set and construction <t< td=""><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></t<>								
in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition in Year Addition Addition in Year Addition in Year Addition in Year Addition Addition Addition in Year Addition in Year Addition in Year Addition			1					
BOX Addessort 1 3 3 5 3 5 <		17010006	1					
B wash colume memory		11010000	1					
BY RECEIVE IF 2000 S 2 3 2 5 BY RECEIVE IT 210004 S 2 3 5 2 5 BY RECEIVE IT 210004 S 2 3 5 2 5 5 2 5 5 5 2 5 5 5 2 5			56	437				
		:BF0000	5	2				
i r x BLOCK ITERATOR i i i i i i i i i i i i i i i i i i i			5	2	2 5			
TABLE ACCESS FULL 13 13 13 143 IP X RECEVE 1000 810 433 2 60 IP X RECEVE 180000 810 433 2 60 IP X RECEVE 180000 810 433 2 60 IP X RECEVE 180001 41 2 2 60 IP X RECEVE 180001 4 2 2 60 IP X RECEVE 1701002 4 2 2 60 IP X RECEVE 14 2 2 60 840 IP X RECEVE 14 2 2 60 840 IP X RECEVE 144 2 2 60 840 IP X RECEVE 144 2 2 60 840 IP X RECEVE 144 22 2 60 144 145 IP X RECEVE 144 22 2 145 2 145 IP X RECEVE 144 22 2 15 15 15 IP X RECEVE 3 2 <td>PX SEND HASH</td> <td>:TQ10004</td> <td>5</td> <td>2</td> <td></td> <td></td> <td></td> <td></td>	PX SEND HASH	:TQ10004	5	2				
TABLE ACCESS FULL 13 3 2 3 2 3 ■ X RECEVE 100 M FULTER USE 180000 850 435 2 60 ■ MASH MUSE 180000 850 435 2 60 2 60 ■ MASH MUSE 180000 850 435 2 60 2 60 ■ MASH MUSE 180001 42 2 60 2 60 2 60 ■ X RECEIVE 4 2 2 4 2 2 60 2 60 ■ X RECEIVE 4 2 2 4 2 60 840 60 ■ X RECEIVE 14 4 2 2 60 840 60			5	2	₂ ₅ ≻ 60			
PX SEND MASH rrq10005 810 435 2 60 D DM FLITER USE 1870000 810 435 2 860 746 D DM FLITER CEARE 1870001 4 2 2 40 746 D DM FLITER CEARE 1870001 4 2 2 40 746 PX SEID MASH 1701002 4 2 2 40 746 PX SEID MASH 1701002 4 2 2 40 746 PX SEID MASH 1701002 4 2 2 40 746 PX SEID MASH 1701002 144 422 2 8840 8400 PX SEID MASH 1701003 144 432 2 135 746 PX SEID MASH 1701000 3 2 3 3 2 3 3 PX RECEIVE 144 432 2 135 15K 15K 15K PX RECEIVE 343 2 3 2 3 3 2 3 3 15K		Т3	5	2	2 5			
BODIN FILTER USE 1870001 810 433 2 840 20 40 20 20 40 20 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 40 20 100 40 20 100 40 20 100 40 20 100 100 40 20				435				
HASH LOUI BUFFEED IBF001 4 2 2 B40 2 M8 D IN FLICE CARATE IBF001 4 2 2 4 4 D IN FLICE CARATE IBF001 4 2 2 4 4 D IN FLICE CARATE IBF001 4 2 2 4 4 D IN FLICE CARATE IF COLOR ITERATOR 4 2 2 4 4 D IN ELOSC ITERATOR 4 2 2 4 4 2 4 D IN ELOSC ITERATOR 4 42 2 44 <					-			
ib D01h FILTER CREATE 186001 4 2 2 4 ib PX RECEIVE if Q10002 4 2 2 4 ib PX RECEIVE if Q10002 4 2 4 2 4 ib PX RECEIVE if Q10002 4 2 4 2 4 ib PX RECEIVE if Q10003 if A4 432 2 840 840 ib PX RECEIVE if A4 148 432 2 840 16 ib PX RECEIVE if A4 148 432 2 15K 2 840 ib PX RECEIVE if A5001 if A4 432 2 15K 2 15K ib PX RECEIVE if A5002 3 2 2 3 2 15K 15K ib PX RECEIVE if PX RECEIVE if PX RECEIVE 3 2 3 2 3 15K ib PX RECEIVE if PX RECEIVE if PX RECEIVE if PX RECEIVE 1 3 2 3 1 1 ib PX RECEIVE if PX RECEIVE if PX RECEIVE		:BF0000		435				
ip XR RECEIVE ip X BRON MASH i7 CQ10002 ip X	A			435				
Image: PX SEND HASH iTQ10002 4 4 2 4 4 2 4 4 2 4 4 2 4		:BF0001	4	2				
PX BLOCK ITERATOR 14 2 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 4 2 840 2 840 PX RECEIVE 114K 432 14K 432 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 840 2 155 2 155 2 155 2 155 2 155 2 155 15K 15K </td <td></td> <td>17010002</td> <td>4</td> <td>2</td> <td></td> <td></td> <td></td> <td></td>		17010002	4	2				
TABLE ACCESS FULL T2 4 2 4 2 840 PX RECEIVE 114K 432 2 840 2 840 PX SEND MASH 17Q1003 114K 432 2 15K 2 840 PX SEND MASH 17Q1003 114K 432 2 15K 2 840 PASH JOIN BUFFERD 114K 432 2 15K 2 840 2 15K PX RECEIVE 16002 3 2 15K 2 3 2 15K 2 15K PX RECEIVE 170000 3 2 3 2 3 2 3 2 3 15K	-	:1010002	4	2				
PX RECEIVE 14k 432 2 840 2 840 PX SEND HASH 170003 14k 432 2 15k 2 840 PX SEND HASH 170001 14k 432 2 15k 2 16k P MASH JOIN BUFFRED 14k 432 2 15k 2 16k 2 16k P MASH JOIN BUFFRED 14k 432 2 15k 2MB 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 2 3 3 2 3 3 2 3 3 2 3 3 2 3 3 2 3 3 2 3 3 2 3 3 2 3 3 3 2 3 3 3 3 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3	_	T2	4	2				
Image: px send hash iTQ10003 14k 432 2 840 1			14К	432				
Image: Doin Filter USE 18F0001 14K 432 2 15K Image: Doin Filter Create 18F0002 3 2 3 2 3 Image: Doin Filter Create 18F0002 3 2 3 2 3 Image: Doin Filter Create 18F0002 3 2 2 3 2 3 Image: Doin Filter Create 18F0002 3 2 2 3 2 3 2 3 2 3 2 3 3 2 3 3 2 3 3 2 3 3 3 2 3 3 3 2 3 3 3 3 3 3 2 3		:TQ10003						
i join filter create iif poo2 iif coo2 iiif c			14К	432				
PX RECEIVE ITQ1000 3 PX SEND HASH ITQ1000 3 PX SEND HASH ITQ1000 TABLE ACCESS FULL T1 PX RECEIVE 343 PX SEND HASH ITQ1001 343x 427 PX SEND HASH ITQ1001 344x 1 1 1 1 1 1 1 1<	HASH JOIN BUFFERED		14K	432	2 15К2МВ			
PX SEND HASH iTQ10000 iTQ10000 PX BLOCK ITERATOR ITA PX BLOCK ITERATOR ITA ITABLE ACCESS FULL ITA <		:BF0002	3	2				
Lip X BLOCK ITERATOR Lip X SELO CK ITERATOR D PX RECEIVE D PX RECEIVE D PX SELO HASH 1 TQ10001 1 TQ10000 1 TQ10000 1 TQ10000 1 TQ10000 1 TQ10000 1 TQ10000 1 TQ100000 1 TQ100000 1 TQ100000 1 TQ1000000 1 TQ			3	2				
L TABLE ACCESS FULL T1 3 2 2 3 PX RECEIVE 343% 427 2 15K PX SEND HASH iTQ1001 343% 427 2 15K D JOIN FILTER USE iBF002 343% 427 2 15K P X BLOCK ITERATOR 343% 427 2 15K		:TQ10000	3	2				
Image: Label ACCESS Full. T1 3 2 3 Image: PX RECEIVE 3434 427 2 15K Image: PX SEND HASH iTQ1001 3434 427 2 15K Image: Distribution Filter Use iBF0002 3434 427 2 15K Image: Distribution Filter Use iBF0002 3434 427 2 15K Image: Distribution Filter Use iBF0002 3434 427 2 15K Image: Distribution Filter Use iBF0002 3434 427 2 15K			3	2	² ³ ≻ 15K			
Impose the problem of the problem		Τ1	3	2	2 3			
b join filter use 18F002 343K 427 2 15K b px block iterator 343K 427 2 15K								
D PX BLOCK ITERATOR 343K 427 2 15K								
		IDFUUGE						
		T4				81		
								10
)) 19:5 30/05/2

Jonathan Lewis © 2011 - 2016 Compare the actual rows with estimates and you can see the "actual rows" figures drop by N/70 *before* each join as each Bloom filter is used.

PX Plans p. 30 / 34

PQ Stats (hash / hash)

DFO_NUMBER	TQ_ID	SERVER_TYPE	INSTANCE	PROCESS	NUM_ROWS	
1	0	Producer	1	P002	3	scan t1
			1	P003	0	pass to 0/1 to build
		Consumer	1	P000	2	
			1	P001	1	return filter (b1)
	1	Producer	1	P002	7297	scan t4 filter (b1)
			1	P003	7405	Pass to 0/1
		Consumer	1	P000	9801	buffer
			1	P001	4901	buffer
	2	Producer	1	P000	4	scan t2
			1	P001	0	pass to 2/3 to build
		Consumer	1	P002	3	
			1	P003	1	return filter (b2)

Jonathan Lewis	The first two steps of the TQ stats show slave set 2 scanning t1 then scanning t4 with	PX Plans
© 2011 - 2016	a Bloom filter - but slave set 1 doesn't join the two rowsources straight away.	p. 31 / 34

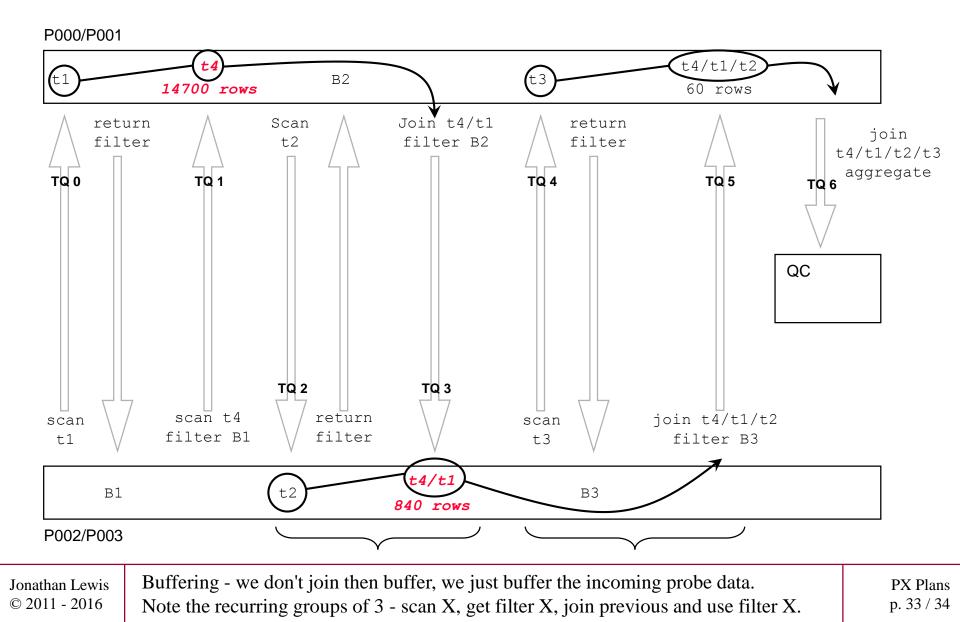
PQ Stats (hash / hash)

DFO_NUMBER	TQ_ID	SERVER_TYPE	INSTANCE	PROCESS	NUM_ROWS	
	3	Producer	1	P000	560	t1/t4 filter (b2)
			1	P001	282	Pass to 2/3
		Consumer	1	P002	632	buffer
			1	P003	210	buffer
	4	Producer	1	P002	5	scan t3
			1	P003	0	pass to $0/1$ to build
		Consumer	1	P000	4	
			1	P001	1	return filter (b3)
	5	Producer	1	P002	45	t2/(t1/t4) filter (b3)
			1	P003	15	
		Consumer	1	P000	48	Pass to 0/1
			1	P001	12	
	6	Producer	1	P000	1	join t3/(t2/(t1/t4)
			1	P001	1	and aggregate results
		Consumer	1	QC	2	

Jonathan Lewis © 2011 - 2016 After sending t2 to slave set 2, slave set 1 joins t1 and t4 and sends the result to slave set 2 - but slave set 2 doesn't join these two rowsources straight away.

PX Plans p. 32 / 34

Graphic (hash / hash)



Observations

- Follow the TQxxyyyy name order within DFO tree
 - "Name" = :TQxxyyy and "TQ" = Qxx,yyyy
- Hash Join *Buffered* may spill the "large table" to disc
 - Use lots of memory and broadcast
- Bloom filters "hide" (in 11g)
 - Look at v\$pq_tqstat, 10046, OEM Monitor (v\$sql_monitor)
- Bloom filter numbering is "wrong"
 - (The same is true of DFO trees)
- Keep an eye on v\$pq_tqstat for uneven distribution
 - But it has many limitations. SQL Monitor is far better if licensed