



How Hardware Performance Translates into Oracle Performance



By Stéphane Faroult

Does more powerful hardware always result in better performance? Not always, argues this article. Using a simple query, the author shows how it performs across various levels of hardware power and how the performance changes by simple tuning, provoking some critical thinking about hardware upgrades.



It's quite common to hear frantic calls for a hardware upgrade when performance becomes a problem. "More memory, more processors, faster processors." However, how much bang will we get for the buck? This is obviously one of those haunting questions. This article will not provide a magical recipe for devising how much hardware power a given application requires. However, it will comment on the execution of a simple query on hardware ranging in price from about USD 12,000 to USD 250,000. A disclaimer needs to be stated here: The figures given hereafter have no pretense to resulting from a careful benchmark. They are, however, actual results of tests carried out on a number of Sun servers in a real, live environment. The main purpose of this article is first of all to provoke some thought about hardware upgrades.

A Simple Test SQL Query

While a proper benchmark can show up hardware performance differences, it was not the intention to set up something complex for our test. There are some "universal queries," usually data dictionary related, which run on about any Oracle database. I wanted something conceptually easy, and yet not too simple a query, so I chose to answer the question "How many tables, bar the tables belonging to SYS, have no primary key?" It is a simple question, and means that we just have to look for the tables for which we find no primary key entry in the dictionary view of constraints.

As is often stated, one must write queries carefully, since most performance

gains come from writing them efficiently. Avoiding full table scans, always using NOT EXISTS instead of NOT IN, etc., are some of the performance tips which any performance-conscious Oracle developer knows. So let's write our nice little query:

```
select count(*)
from dba_tables t
where not exists
(select null
 from dba_constraints c
 where c.owner = t.owner
 and c.table_name = t.table_name
 and c.constraint_type = 'P')
and t.owner != 'SYS';
```

We roughly expect that such a query is, performance-wise, mostly dependent on the number of tables inside the database. We selected a number of databases on servers that are vastly different in speed and processor count, where the number of tables was of the order of magnitude of 2,000 (1,450 to 2,920 to be precise). In other words, the number of tables was within a defined range while not being exactly the same.

Here are the results:

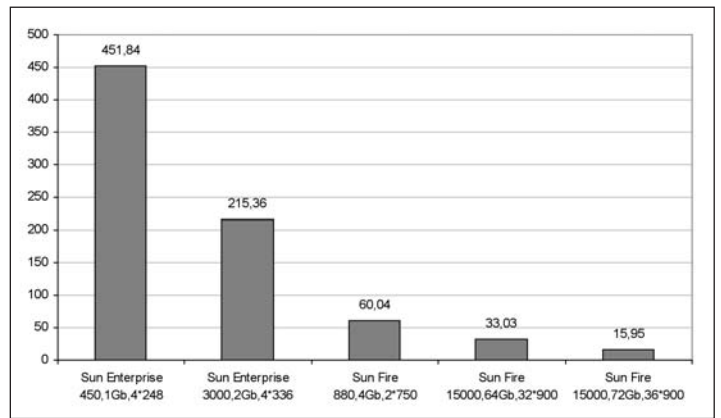


Table 1 : Counting tables with no PK in a 2,000-table database

[Editor's note: For non-European readers, the values '451,84' should be read as '451.84' and so on]

On the slowest hardware, a Sun Enterprise 450 with one Gigabyte of memory and 4 CPUs clocking at a leisurely 248 MHz, the query took 28 times longer to tell us that 995 tables (out of 1511) had no primary key than on the top of the range Sun Fire 15000 server with 36 CPUs running at 900 MHz (on which we had only 673 such tables out of 1787).

Our results seem to show that the most sensitive factor seems to be processor speed. Although some other factors may have been at work, at the top of the range, the number of processors may have also made a difference. In this special case, however, even the number of processors appears to be irrelevant, when you consider the quite decent performance of the only bi-processor computer in our sample. The results, of course, would be different in a heavily concurrent environment, or if the query were run concurrently.

Most interestingly, the impressive 28:1 performance ratio between our top performing server and our worst performing server is indeed a tad better than the 21:1 price ratio, not considering software licence costs. So, what is our conclusion? The investment delivered the performance we were looking for, didn't it? Well... Maybe not! We will address this question below.

continued on page 16

A Simple Test SQL Query, Revisited

Let us reconsider the result. 452 seconds (or 7.5 minutes) to access the DBA_TABLES view consisting of about 2000 rows and the DBA_CONSTRAINTS view that usually contains 4 or 5 times the row count of the former is not exactly glorious performance, even on modest hardware. However, it is possibly less pathetic than 16 seconds on a 36 * 900 MHz processors server. This then begs the question: Are we starting from a good premise?

Quite obviously, improving the speed of a dictionary query by adding indexes is something we cannot do. However, it's often interesting, even without jumping from the start on /*+ HINTS */ to consider alternative ways to write it. We could use the dreadful, definitely not to be used, NOT IN clause. So let us recode the query as shown below:

```
select count(*)
from dba_tables t
where (t.owner, t.table_name)
not in
(select c.owner, c.table_name
from dba_constraints c
where c.constraint_type = 'P')
and t.owner != 'SYS';
```

There is also another very weird way to write a query, which involves something you have to be very careful with—an outer join. The idea is to select from the outer join those rows which have no match with an IS NULL on some of the outer-joined columns. This is another condition of bad repute, but we are still experimenting. As well, let us throw in an in-line view, just to make things more complicated. The rewritten query is shown below:

```
select count(*)
from dba_tables t,
(select c.owner, c.table_name
from dba_constraints c
where c.constraint_type = 'P') c
where t.owner != 'SYS'
and t.owner = c.owner (+)
and t.table_name = c.table_name (+)
and c.table_name is null;
```

There is yet another option, which involves set operators. These are usually favourably spoken of, so let's go for a MINUS.

```
select count(*)
from (select owner, table_name
from dba_tables
where owner != 'SYS'
minus
select owner, table_name
from dba_constraints
where constraint_type = 'P');
```

There is an obvious flaw in this query, which we can see at once. That is, since none of the filtering criteria is very selective or likely to be indexed, we'll only have full table scans here (and DBA_CONSTRAINTS, which contains all constraints including NOT NULL, is much bigger than DBA_TABLE). But let's use it nonetheless.

So what did running the various SQLs show? The answer is shown in the graph below.

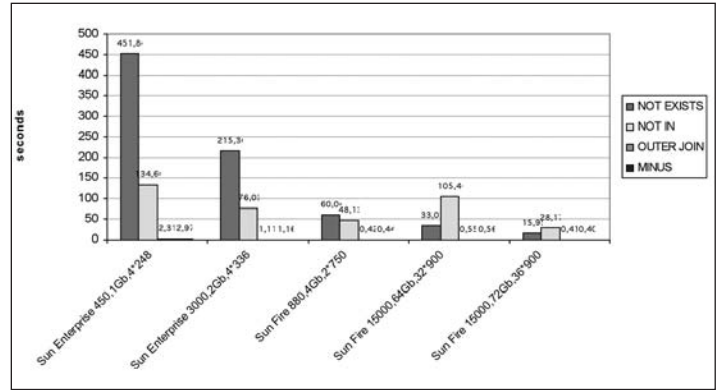


Table 2 : Counting tables with no PK in a 2,000-table database—the full picture

The results are more than surprising. The weird query and the double-full-scan MINUS perform rather well, as seen above. Even the dreaded NOT IN, except on the two most powerful computers, usually massively outperforms the NOT EXISTS. It gets more interesting to see the ratios. On the same machine, the ratio between the worst performing query and the best performing query is usually between 150 and 200:1, an order of magnitude greater than the results we previously saw between the cheapest and most expensive hardware.

There is something which is slightly distressing though. The best query on the lowly Sun Enterprise 450 runs more than 5 times faster than the worst performing query on the all-powerful Sun Fire 15000 which has 9 times more processors, each of which are 3 times faster!

Another worrying thought is that although we had a good 28:1 ratio between the most expensive and cheapest hardware for the worst performing query, this ratio is down to 7.5:1 with the best performing one, which of course makes the invoice look perhaps a bit less justified.

Coup de Grace

Call me a purist, but I have always been uncomfortable working with dictionary views. Those views bring back a lot of information with which, in this particular case, we don't have anything to do with. We can write something returning exactly the same result by hitting only the SYS tables we really need. The query is shown below and is built along the same lines as our fastest query so far:

```
select count(*)
from sys.obj$ o,
(select obj#
from sys.cdef$
where type# = 2) c
where c.obj# (+) = o.obj#
and o.type# = 2
and o.owner# != 0
and c.obj# is null;
```

I have only run it on the slowest E450 server. It took just 0.26 seconds, which brings our "slowest query to fastest query" ratio to 1737:1 on this server. I doubt it would run much faster on the top of the line hardware which costs USD 250,000. We have also shown only Sun servers in this contrived benchmark in order to simplify the comparison – the results would have been similar on any other platform.

Conclusion

My present purpose is to reduce the army's size and relieve some of the commanders, to punish openly those who deserve it while reflecting on our own mistakes, and to revise our tactics for the future. Unless we do this, simply increasing our forces won't help.

Three Kingdoms (attributed to Luo Guanzhong), Chapter XCVI

I have tried to demonstrate several points in this paper, one of which is that hasty generalisations are not necessarily very reliable. As well, concluding in a generalized fashion from this paper that any Datawarehouse or telco billing application can be run with lightning-speed on any decommissioned 5-year-old PC would definitely be getting the message wrong.

As the disclaimer stated, this is no rigorous benchmark, just an eye-opener.

Something which I find striking is the magnitude of difference in raw power, which shines bright when the queries are poorly written, dwindles when they are well written. It is beyond the purpose of this paper to explain how much logical I/Os hammer your CPU and how reducing them should be a prime target. Others, such as Cary Millsap, Tim Gorman, Jonathan Lewis, Connor McDonald and Wolfgang Breitling have written excellent technical papers on such questions. But the truth is that powerful hardware will not speed up a good program by as much as it will speed up a bad one. The bad program will still be less efficient than a good program running on cheaper hardware. The purpose of powerful hardware should not be to compensate for the inadequacies and inefficiencies in the code, but rather to allow more processing in parallel, providing for breadth rather than depth of computing power. Many a times, you will find that spending some time, money and resources on tuning may give you a much better payback than spending much more on "powerful" hardware. We hope this article helped you start thinking in the right direction, and ask probing questions about the quality and efficiency of your code before jumping to the conclusion that the 'latest and greatest' hardware will solved your performance problems.



About the Author

Stéphane Faroult is a graduate from the Ecole Centrale de Paris, one of the top French scientific Grandes Ecoles. He joined Oracle France in their early days after a brief spell with IBM and a bout of teaching at the University of Ottawa and soon developed an interest in performance and tuning topics. After leaving Oracle in 1988 and a short incursion in the field of operational research, he went into Oracle consulting, mostly for big companies in almost all sectors of activity and has gone on since, although under different labels. He co-founded Oriole in 1998 with British partners and can be contacted at sfaroult@roughsea.com.