Tuning Guidelines

Database Tuning

If a performance problem is to occur anywhere in an n-tier application, it is most likely to occur in the database. Evaluation of the database is not really that difficult and comes down to understanding just a couple of key concepts.

Physical I/O

This must be minimized as much as possible. Physical I/O to disk is not only slow, it also consumes CPU. There are many ways of reducing Physical I/O, traditionally this would involve:

* Ensuring that the database has sufficient memory available for it to be able to store commonly accessed data in a data cache.
* Ensuring that data retrieving queries efficiently find target data without having to trawl through large tables.

CPU Utilization

While a low CPU utilization for a database server may appear to be good, it could by symptomatic of a poorly configured database. If the database is I/O bound, or lacks a sufficiently tuned configuration, it is possible that the database is not unable to service more requests for data but is not receiving those requests very quickly, i.e., those requests for data are queuing.

In a well configured database server, busy periods will see CPU utilization around 80% with occasional peaks in workload taking the CPU utilization closer to 100%

CPU utilization can be reduced by investigating the following areas;

As a request for data arrives at the database, the SQL query must be parsed. This basically means that Oracle investigates the statement and decides how it will access the database tables so as to satisfy the query request. If the query has already been executed, it may still be in the area of the cache where queries are stored. Reusing a query which has already been parsed and is in the cache uses much less CPU than parsing a query for the first time. To ensure this reduction in CPU utilization, the performance tester should ensure two things:

- The cache is large enough to store the required number of queries.
- Bind variables should be used so that the where clause of the query is not affected by different data variables; e.g. where name =b1: rather than where name = smith.

Once the query has been parsed, execution of the query will take place. It is essential that the database is able to find the requested data simply and easily. The access method that will be used to locate and retrieve the data is determined when the query is parsed. By using explain plan, the performance tester can determine what the access path is. An efficient query will require only a small number of rows to be read. Hopefully, these rows will be located in the databases buffer cache rather than on disk. An inefficient query will require a large number of rows to be accessed. Even if the much larger number of rows reside in the buffer cache, it still requires a certain amount of CPU to locate, retrieve and possibly sort the data

Load Testing- Bottlenecks

Bottlenecks that you will not encounter in production

The performance and load test environments tend to be smaller versions of the production environment. These differences can include:

- Less memory
- Fewer and smaller physical CPU’ s
- Fewer and less efficient disk arrays
- Single or fewer instances of servers, e.g. 2 database servers in performance test, but 3 database servers in production or even no clustering at all

There can be other differences as well. System software & hardware versions and specifications can vary. Data can be older and less comprehensive than in production. There also may be different authentication, firewall or load balancing configurations in place. Data volumes may be less, the O/S may be 32bit instead of 64bit or vice versa.

The performance test itself is full of compromise. For instance:

- Normally only a subset of functions are automated. The path through the function may vary slightly, but generally the same steps are repeated each iteration
- Data can be mass produced in a uniform manner, affecting the way data is stored and accessed on the database. Some database tables can contain too little data, others too much The behavior of users can be unrealistic, for instance, if a requisition is raised, it is not generally fulfilled in reality until some days down the track. In a performance test, it may be 5 minutes later.
- Workload being processed varies in the normal course of events; during a performance test it can remain uniform

This can create many problems for the performance tester. Understanding the differences between any performance test environment and the production environment is essential. This can help detect and understand an artificial bottleneck. This is something quite different from a performance bottleneck.

An artificial bottleneck is essentially a performance problem that is a direct result of a difference between the production environment or workload and the performance test environment or workload. It is not a performance bottleneck. A performance bottleneck is something that could or is happening in production.

When a performance bottleneck is found, the performance tester must investigate the symptoms in an attempt to try and pinpoint the cause of the issue. Care must be taken to distinguish between a genuine performance bottleneck and an artificial bottleneck brought about purely because of differences in performance test compared to production.

Examples of artificial bottlenecks include:

- Database locking - Performance testing results in a subset of functionality being automated. The entire workload is then spread across this subset of automation resulting in some functions been driven at a higher frequency during performance test than would be seen in production. This can result in database locking which would not occur in production. The solution? The problem function should be driven no higher than the maximum peak workload expected in production.
- Poor response times and excessive database physical or logical I/O –

If a large amount of data has been recently added to the database via a data creation exercise, database tables can become disorganized, resulting in inefficient use of indexes.

Additionally, if performance testing started with the database tables nearly empty of data, the optimizer can incorrectly decide that index use is not required. The solution? Run a databasereorganization and refresh optimizer statistics at regular intervals if large amounts of data are being added to the database.

• Poor response times and excessive database physical I/O - The assumption here is that the database buffer pool is smaller in performance test than it is in production due to a shortage of memory in the performance test environment. Unfortunately, this is a case of poor performance of some users impacting the performance of all users. The solution? Once the problem function (or functions) is identified, attempt to decrease the workload for those functions to minimise physical I/O. If this is not possible, it may be time for a memory upgrade of the server. Memory upgrades are usually relatively straightforward in terms cost and time.

• Memory leak - This is a terrible term describing how memory is gradually consumed over time inferring that at some point, there will be no free physical memory available on the server. Performance test environments often differ from production environments in terms of house keeping. In production, the application could for example be refreshed each night, in performance test it may have been nine weeks since it was last refreshed. The amount of memory in production is often substantially more than in performance test. The solution? Base memory leak calculations on the production memory availability with reference to the production house keeping regime. Remember that once physical memory is fully consumed, virtual memory is still available so it' s not the end of the world as we know it.

Last but not least, when a genuine problem is found, the initial reaction from the developers, the DBA's, the architects, the project management, pretty much everyone really is that the problem observed is not a real problem, it is an artifact of the test tool. There seems to be a dis connect from reality, a lack of understanding that the behavior of the application changes depending on the workload. Every performance tester out there should know what I am talking about. In fact, please quote this article when discussing these issues with the project. It is down to the performance tester to come up with some approach that will satisfy everyone concerned that the problem being observed is or is not due to the automated test tool, i.e. is not an artificial bottleneck. This is a scientific approach, develop a theory, then design a test that should prove or disprove that theory. It does not really matter if the theory is right or wrong, every time you devise a new test and execute it, you learn something new about the behavior of the application. The performance tester has to work hard to gain the trust of the project and through thorough performance analysis, demonstrate that a performance problem is just that, a performance bottleneck that will cause an impact in production.

An example of a test tool generated performance issue:

- Ramping up the workload too quickly. If virtual users start too quickly, the application struggles to open connections and server requests. For most applications, this is abnormal behavior that you would not normally observe in a production environment.
- Configuring the workload in such a way that it is spread unevenly across the period of an hour, for instance users with just 2 iterations to execute waiting for excessive amount of time between iterations causing a quiet patch in the middle of a Load test.
- Restarting the application just before performance testing starts. While this can help to maintain consistency of results, throwing a large number of users at an application where data cache is not populated and software and hardware connections are not established can cause an unrealistic spike in performance as well as much longer than expected response times. In reality, application starts normally happen around 05:00 in the morning when few users are around. The first few users of the day will experience longer responses but these will be limited. By the time a much larger workload is starting to ramp up (such as in a performance test), data cache, threads and connections are already open and available and will not cause users these long response times.
- Putting a think time inside the start and end transaction markers denoting response time so that the recorded response time for a transaction is much longer than it actually is.

Automated Test Script Correlation

A Correlation is a Connection or Association

For automated test scripts, in practical terms, it means that the application has associated some information or data which is specific to a particular user session. A context has been established.
When a user logs into an application, the application may assign a session id to a particular user. The session id is managed by the server and the client so that the user does not need to know anything about the session id. When the server receives a message, it uses the session id to identify which user it has received a message from. Knowing who the user is can be used for many activities such as:

1. Ensuring the user is allowed to access a screen or data item
2. Identify which data is pertinent to a particular user
3. Log an audit trail of data and application usage

When generating an automated test script, the first step is to record the automation. This collects all relevant information exchanged between the client and the server, including session ids. Interestingly, by default, most of the major test tools encrypt the password that is captured during a recording session. For example, with Loadrunner, 'password' could become '4a49e9ac50ecbc528a311ce9'

If the performance tester were to attempt to play back the recorded script, it would most likely fail. The recorded session id would no longer be valid. When a request was received by the server, it would determine that the session id had expired and was no longer valid and therefore would decline the request.

In order for the script to work, the performance tester needs to correlate the session id. The first thing to do is to find the session id in the log. An example of what a sessionid looks like follows:

"98d3HCnGlg2RswJhW9HlbyCn91YWnCQ12DsR4yNjnYv5PsY13thL!-451350722!-1247883591"

The recorded value will differ from the value returned to the automated test script on first playback. Hopefully, the length of the session id will remain constant. It may take some searching to find, but careful study of a detailed log will eventually yield the session id.

The session id then needs to be captured. All of the major test tools allow for capturing of returned strings as part of correlation. An example below shows part of a buffer trace, and the Loadrunner command to capture the session id.

```
web_reg_save_param("Sessionid”  ,
"LB/IC=SESSIONID=",
"RB/IC=\"",
"Ord=1",
"Search=body",
LAST);
```

Loadrunner Analysis Traffic Feature
Vugen Script Generated from a Trace File
Where the application front end cannot be recorded, Vugen can analyse server traffic files to create automated test scripts using the Analyze Traffic feature. This will create what is known as a Server Traffic Script.

The Java protocol is a case in point. Loadrunner Java scripts for load testing can only be played back. They cannot be recorded. The question is, will the analyse traffic feature help performance tester to develop a working script?

As usual, the Vugen user guide makes it sound easy. Step 1: generate a capture file. Is this as easy as it sounds, or is it a bit more complicated?

Apparently the capture file is a trace that contains a log of TCP network traffic. I hate the term ‘sniffer application’ , it is meaningless. A dump of the network traffic can be obtained by a network tracing tool. My favourite tool is Ethereal. In mid 2006 after a long and bitter dispute involving the trademark, Ethereal was re-named Wireshark, which in my opinion is a much better name any way. Wireshark is a free tool and I know lots of network people who use this tool on a regular basis. Take some time to get to know Wireshark, with a bit a familiarisation you can figure out how to generate a network trace.

If you do not want to use Wireshark, then Vugen has a capture facility that can be used. To do this, a command is issued (from the command line). There are separate commands for different platforms, e.g. for windows, the command is lrtcpdump.exe whereas for HP' s version of Unix, the command is lrtcpdump.hp9.

Now that we have a capture file all we have to do is generate a Loadrunner script. It is at this point in the process that I had very little confidence that the capture file would be valid. Still, all you can do is give it a go. As I started up Vugen the instructions indicated that after processing the capture file I would have either a web services or web soap functions generated. So, after all that effort, I realise that this is just another feature aimed at web applications. Sorry Java or some other bespoke application, you will have to wait another day.

System Tuning Services
Expert Advice in System Tuning
Bottlenecks can occur during performance testing and load testing or performance modelling exercises. They can also of course turn up unannounced in your production environment. When something breaks, or response times suddenly elongate in your production environment, it is natural to try and work out what has changed.
What has been introduced to the application so that performance is suffering and your users are complaining? Maybe there has been a code drop? Maybe a configuration has been amended? This can be a good approach and reversing the change could resolve the performance issues. This is not always the case.
Our performance experts are more interested in attempting to identify where the bottleneck is and what is causing it.

By stepping logically through clues, the performance expert will be able to narrow down the area causing the problem.

Our company has the expertise at hand to investigate and diagnose performance problems quickly and efficiently.

By providing database, web and application servers tuning services, we are able to look at the application as a whole.

Performance tuning goes hand in hand with performance testing and performance modelling. Using targeted testing and modelling procedures, and extensive monitoring techniques, bottlenecks within your application can readily be identified and resolved. By conducting an analysis of the results, points of tuning can be determined, and specific tuning advice can be provided.

In our experience, databases are the single biggest cause of performance issues for an application. By a combination of database configuration changes coupled with tuning of application queries, significant improvements in database performance can be obtained. Please see our knowledge base article on database performance issues.