ORACLE
JD EDWARDS
ENTERPRISEONE

Oracle JD Edwards EnterpriseOne Batch Scalability Tuning

Parallelizing Batch Jobs as Multiple Processes with Required Oracle Database Tuning for Maximum Throughput

ORACLE TECHNICAL BRIEF | SEPTEMBER 2014

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remain at the sole discretion of Oracle.

Table of Contents

Executive Summary

When conducting day-in-the life capacity tests on some new hardware with JD Edwards EnterpriseOne, it was discovered that certain UBEs did not increase in performance linearly as additional UBE processes were added to parallelize the processing of large quantities of data. A project to look into Batch Scalability was then initiated to determine why, and the findings are shared in this technical brief. Ultimately, our team discovered that the database engine was not properly tuned to handle the extreme load that multiple batch processes put on the system. This technical brief describes the performance analysis methodology and tools used, and the actual Oracle database tuning that was applied to our system to achieve performance scalability.

Introduction

Oracle's JD Edwards EnterpriseOne is an integrated applications suite of comprehensive enterprise resource planning (ERP) software that combines business value, standards-based technology, and deep industry experience into a business solution. The JD Edwards solution architecture can exist on multiple platforms and on multiple database architectures. This document describes a methodology for performance analysis and tuning of a system and database for use with the JD Edwards EnterpriseOne Batch Engine when processing multiple batch jobs which parallelize a set of application logic.

Test Configuration

Machines and Platforms

The servers used were all hosted as VMs on a Sun X3-2L system.

Enterprise Server:

- VM on Sun X3-2L
- 8 VCPUs x Intel(R) Xeon(R) CPU E5-2690 @ 2.90 GHz
- 32 GB RAM
- Oracle Enterprise Linux 5.6

Database Server:

- VM on Sun X3-2L
- 4 VCPUs x Intel(R) Xeon(R) CPU E5-2690 @ 2.90 GHz
- 32 GB RAM
- Oracle Enterprise Linux 5.6

Test Controller:

- All tests were submitted on the Enterprise Server via a script invoking runubexml to control data selection.

Software

JD Edwards EnterpriseOne 9.1 with Tools 9.1 Update 4

Performance Measurement Techniques

The following measurements were taken:

Enterprise Server

- **CPU usage** for the duration of the test via `mpstat` and `iostat`
- **Virtual memory** usage for the duration of the test via `vmstat`
- **Disk subsystem** utilization for the duration of the test via `iostat`

Database Server

- **CPU usage** for the duration of the test via `mpstat` and `iostat`
- **Virtual memory** usage for the duration of the test via `vmstat`
- **Disk subsystem** utilization for the duration of the test via `iostat`
- **Enterprise Manager DBConsole** to determine database performance

Use Cases

The following EnterpriseOne Batch jobs were tested: **R09801** - GL Post, **R42800** - Sales Order update, **R31410** - Work Order processing, **R3483** - MRP, and **R43500** - PO Print.

Note that since all the results were very similar for the reports, this tuning example will focus *exclusively* on **R09801** – GL Post and its use cases.

Use Cases #1-4 R09801 GL Post

Use Case #1 GL Post – One job processing 300,000 line items of GL data

» Steps to execute:

- Prepare one runubexml input file for R09081, version PERFTEST1, with 300,000 unique line items of GL data in the data selection.
- Run one instance of R09801, version PERFTEST1, via runubexml with the above XML file.

Use Case #2 GL Post – One job processing 600,000 line items of GL data

» Steps to execute:

- Prepare one runubexml input file for R09081, version PERFTEST1, with 600,000 unique line items of GL data in the data selection.
- Run one instance of R09801, version PERFTEST1, via runubexml with the above XML file.

Use Case #3 GL Post – Five jobs processing 3 million line items of GL data (5 x Use Case 2)

» Steps to execute:

- Prepare five runubexml input files for R09081, version PERFTEST1, each with 600,000 non-overlapping unique line items of GL data in the data selection.
- Run one instance of R09801, version PERFTEST1, via runubexml with the above XML file.

Use Case #4 GL Post – 10 jobs processing 3 million line items of GL data (10 x Use Case 1)

» Steps to execute:

- Prepare 10 runubexml input files for R09081, version PERFTEST1, each with 300,000 non-overlapping unique line items of GL data in the data selection.
- Run one instance of R09801, version PERFTEST1, via runubexml with the above XML file.

Test Scenarios and Design

Each use case was kicked off from a script which utilized runubexml to submit the required number of jobs and the corresponding data selection. Metrics were then collected from the system to determine if any performance bottleneck was occurring. When a performance bottleneck was detected, hypotheses were developed to decide what could be done to resolve the bottleneck, and then the scenario was retested. When a change resulted in no improvement, the change was undone, and the next hypothesis was pursued until the bottleneck was found.

Test Validation

Multiple levels of validation assured that the tests completed correctly, performed the intended tasks, and executed all code paths:

- No unexpected or unexplained messages in any jde.log files on the JD Edwards Enterprise Server or JD Edwards JAS / HTML server.
- SQL verification that the expected number of records were written to the database following completion of tests.
- No failed business functions.

The database was refreshed prior to running each test.

Test data

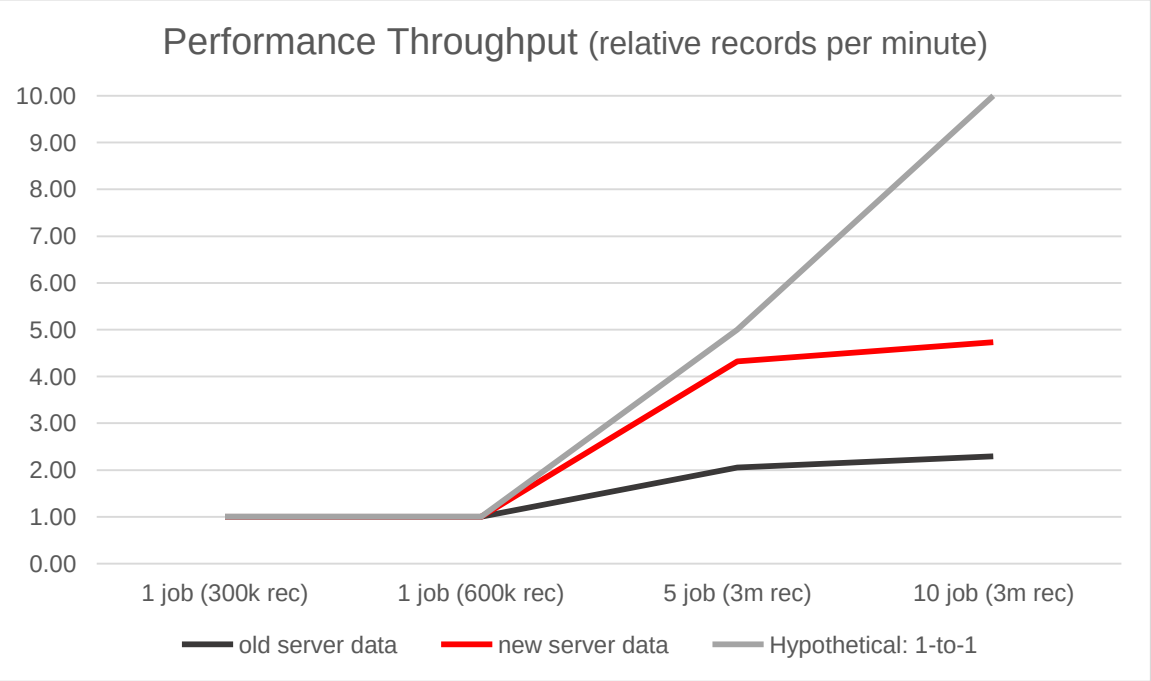
The data for this benchmark was as follows:

- 300,000 GL documents each with 10 line items totaling 3,000,000 rows of GL data for R09801 - GL Post to process.

Test Run Iteration

Initial Run: Duplicating the issue

To determine that our system was able to duplicate the issue we had seen in the previously reported case, we ran each of our test cases and compared our results to the set of results from the previous case.



The data are presented in units of relative records per minute, in comparison to the number of GL Post records per minute throughput achieved by an isolated single GL Post job processing on the server.

The data points furthest to the left (one job, 300,000 records) are Use Case 1, on the middle left, Use Case 2 (one job, 600,000 records), on the middle right we have Use Case 3 (five jobs, 3 million total records), and finally on the far right we have Use Case 4 (10 jobs, 3 million total records).

The three series lines are:

- In light gray, we have the hypothetically perfect case of one-to-one scaling as processes are added. As you can see, this means that, hypothetically, going from one to five processes would ideally provide five times as many records-per-minute of processing performance, while ten jobs would ideally achieve ten times the throughput. *(Please note that, while ideal, in a real system as processes that consume common resources are added perfect one-to-one scalability can never be achieved due to operating system, network, disk, and other process contention throughout the OS and involved databases. However, we strive to come as close as technically possible.)*
- In dark grey are the previous results from another older system, which show that when going from one to five processes, only about two times the throughput was achieved, and when going to ten processes, only about 2.3 times the performance was achieved.
- The current baseline results are in red, demonstrating duplication of the problem. We can see that when going from one to five processes, only about 4.3 times the throughput was achieved, and when going to ten processes, only about 4.7 times the performance was achieved resulting in more or less flat

performance when adding additional processes. Note, however, that in our five-job case we came within 86% of ideal one-to-one performance, which is considered acceptable.

Performance Notes

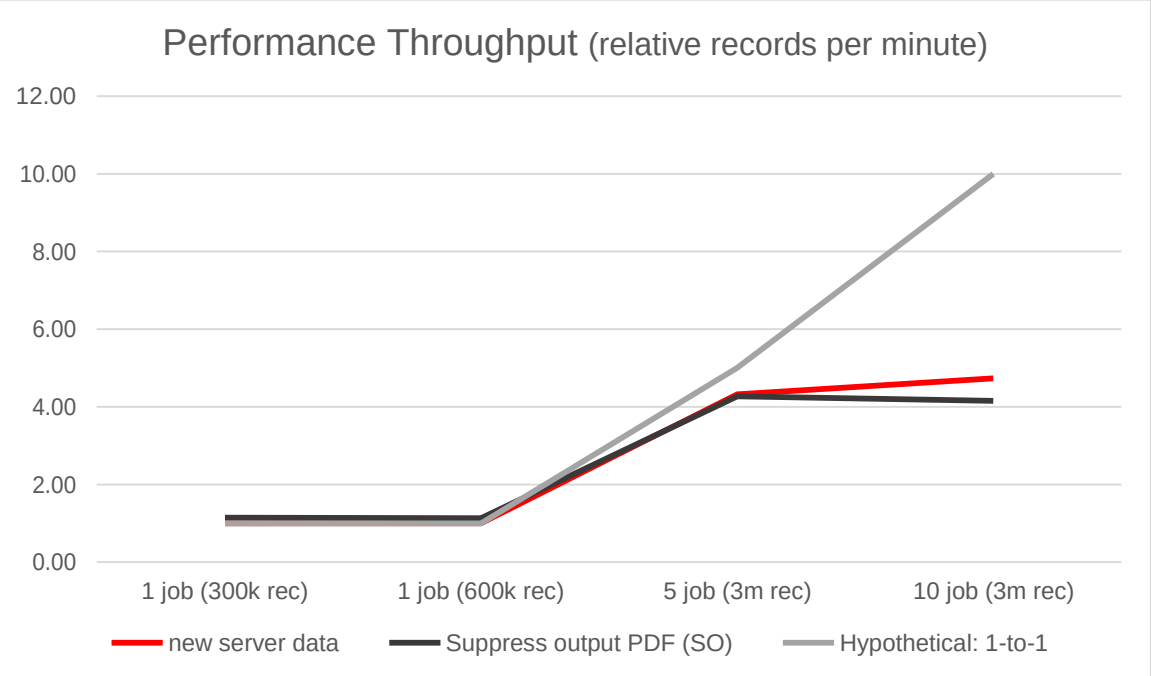
- CPU Activity on the Enterprise and Database server were quite low (under 30%).
- Network activity did not appear to be overly high, but we were concerned and ran a network throughput test demonstrating that these VMs had a 5Gb/sec data throughput and we were not even using 5% of that when running the 10 job test.
- Disk activity seemed somewhat high, although we did not have a good way to determine the maximum throughput of our Xen Hypervisor virtual disk system.

Hypothesis: Disk IO Bottleneck

Our first thought was that our disk subsystems were showing a lot of activity based on the previous measurements, and we were concerned that they were a main part of our bottleneck.

The GL Post UBE produces about 26,500 pages of PDF output per 300,000 records processed, resulting in 28MB PDF files. For this particular UBE, our call-graph analysis tools led us to realize that 25% of the enterprise server's processing time was spent in producing this PDF output outside of business logic time. Further, we realized that PDF generation requires disk temp space roughly equivalent to two times the PDF final size, meaning that for 10 GL Post jobs, we were using almost 1GB of disk IO throughput in file creation.

To determine how much faster we could go if GL Post had output suppressed, we turned off the PDF output engine and re-ran the 10-job use case.



We were surprised by this result: turning off the PDF output made the performance of the single job cases increase by 15%, yet both the five and ten job cases were worse, and the degradation was greater as the number of processes used to parallelize our GL Post was increased.

We decided that whatever our real issue was, suppressing the output had increased the contention on the bottleneck so we would leave it in place as we continued our analysis.

Performance Notes

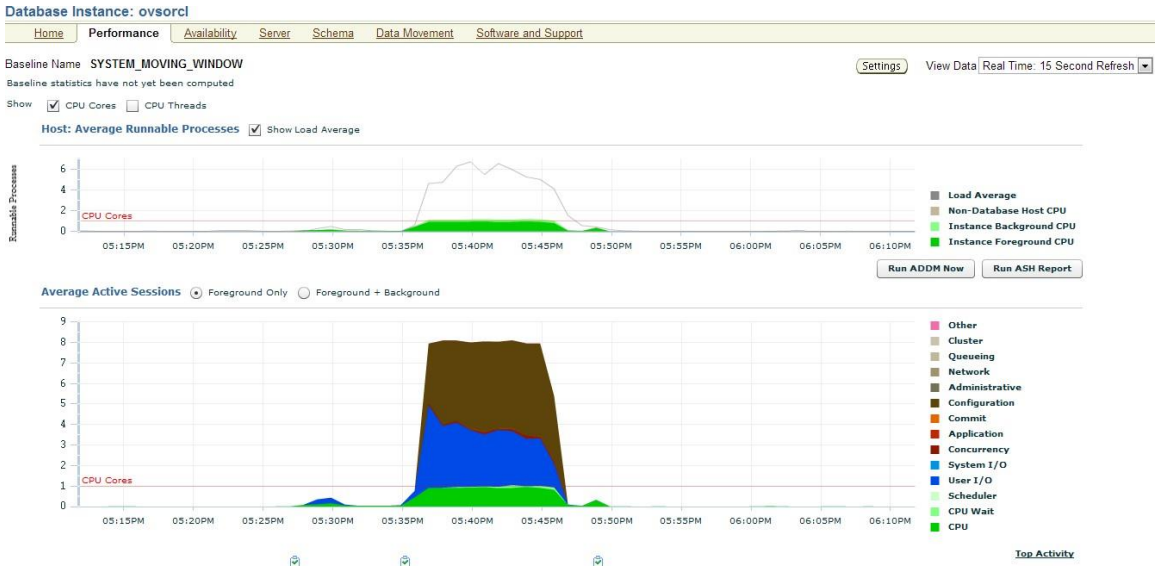
- CPU Activity on the Enterprise and Database server were still quite low (under 30%).
- Network activity did not appear to be overly high, but we were concerned and ran a network throughput test demonstrating these VMs had a 5Gb/sec data throughput and we were not even using 5% of that when running the 10 job test.
- Disk activity was fairly low on both servers after the output was turned off**.**

Hypothesis: Database Contention

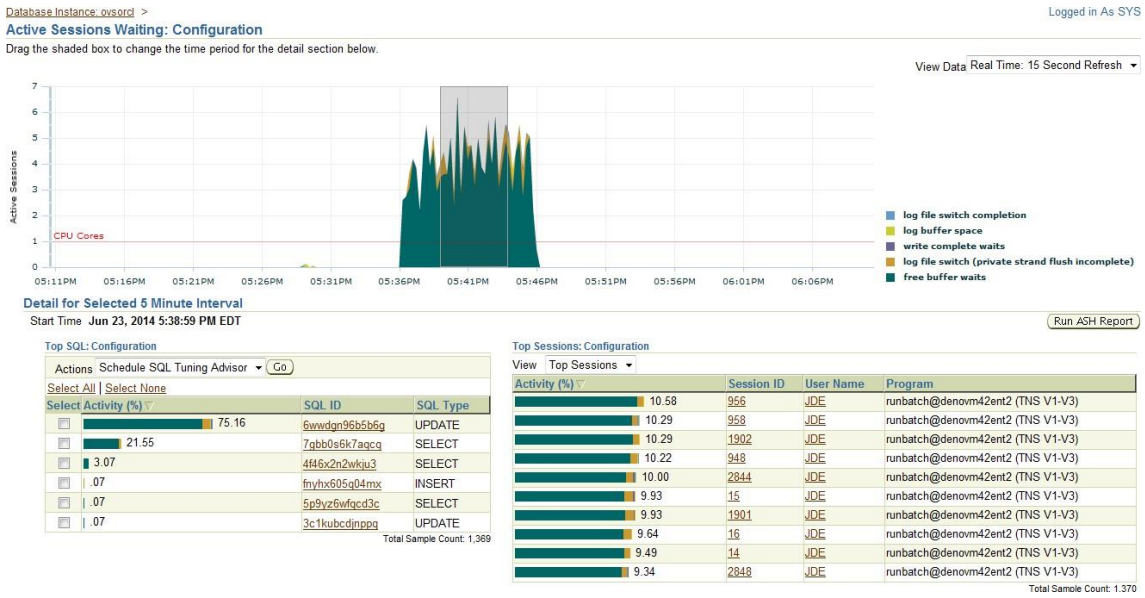
Additional DB Metrics Collected

At this point, we enabled Oracle Enterprise Manager Database Console on our database server, and also, in the "Management Pack Access" setup for Database Console, we enabled the "Database Diagnostic Pack" and the "Database Tuning Pack" in order to analyze what our performance issues might be in the database.

Using use case #4, where we run ten jobs in parallel, we saw the following DB console results:



In this graph from the Database Diagnostic Pack, green is CPU time, blue is IO time, and brown is "configuration time." A well-tuned database should not have any brown – this is an indicator that we have an issue in our database configuration. So, we clicked on the brown area to go to our next graph.



In this graph, the Database Diagnostic Pack shows that the majority of this time is spent on “free buffer waits”.

Once again, we clicked through on the first process on the bottom right to see:

Our “free buffer wait” problem is primarily caused by waiting for “log file switch completion”.

The implication here is that our redo logs, which are the backlog of work for the dbwriter processes used by the database, are filling up faster than the dbwriter processes can commit the updated data to the database.

At this point, we looked at our database and discovered we had three 100MB redo logs, and one dbwriter process.

Looking at the database statistics for the key UPDATE statement used by the GL Post report, we were able to see that 55% of our time was spent waiting for things unrelated to the actual work we would like to accomplish.

At this point, having guidance for tuning our database from our analysis, we spent several iterations tuning the database to improve redo log and dbwriter performance.

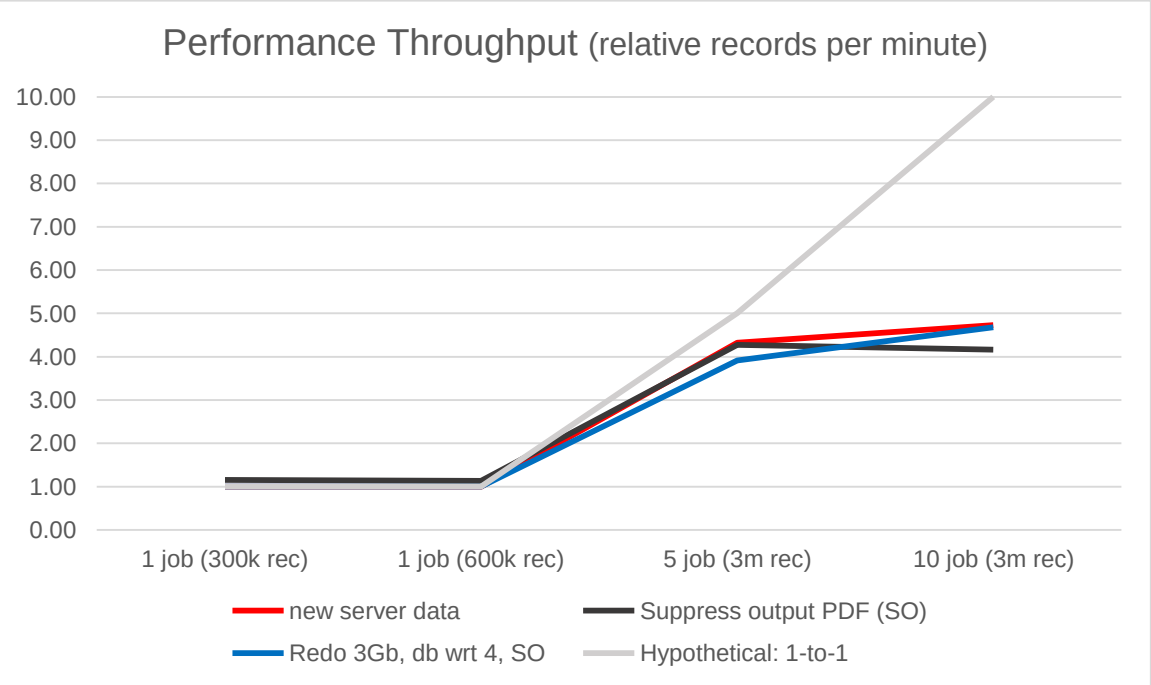
Results

Iteration 1. Our first change was to increase the size of each of the redo logs to 250MB and to increase the number of redo logs to six (1.5GB total). Re-running use case #4, we had essentially the same results reported above.

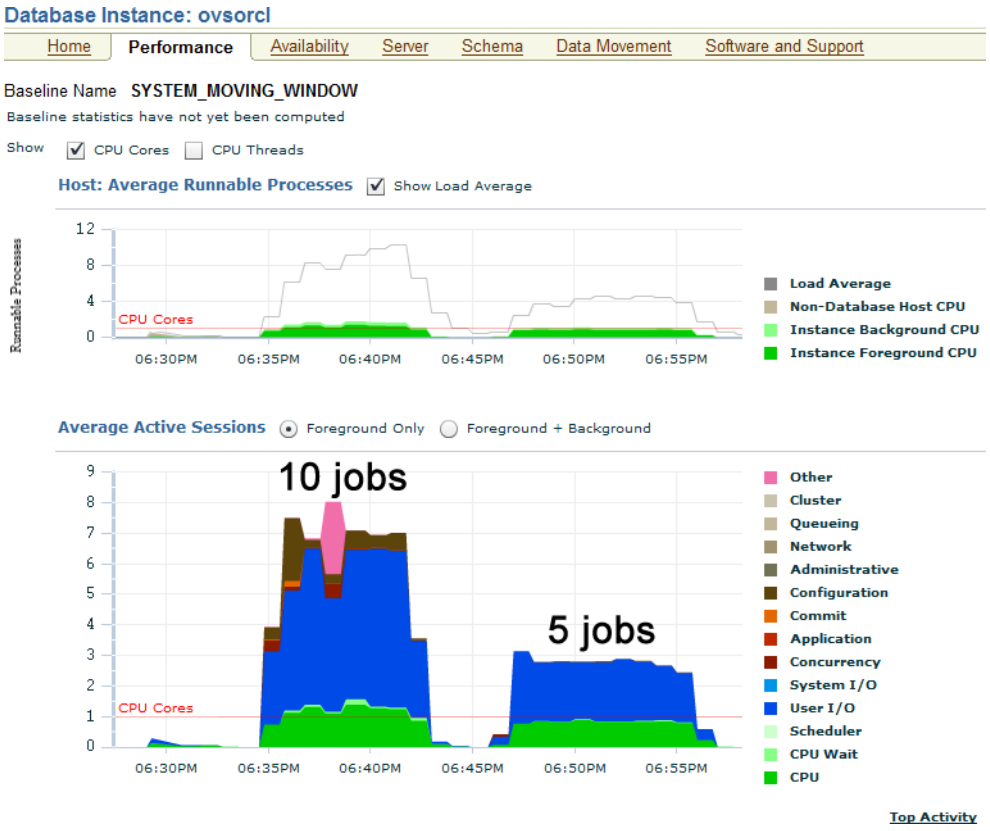
Iteration 2. Next, we increased the redo log size to 500 MB while keeping the number of redo logs at six (3GB total), and we also increased the number of dbwriter processes to two. Re-running use case #4, the results were a little better, but we still performed worse with 10 jobs than with five when measuring total throughput, and we still had the same waits present in the dbconsole performance monitor.

Iteration 3. Our third iteration kept the six redo logs sized at 500MB each and increased the number of dbwriter processes to four. We had, through a dbconsole warning, also discovered that our index tablespace for our business data was 82% full before running our test, and the number of inserts performed put pressure on our index tablespace during the run. Before re-running, we increased the size of our business data index tablespace to the point where it was under 75% full (72% in this case) to diminish that pressure.

Re-running use case #4, we finally see an improvement at 10 jobs in parallel, but ONLY back to the level we had when we were running without output suppressed (graph below).

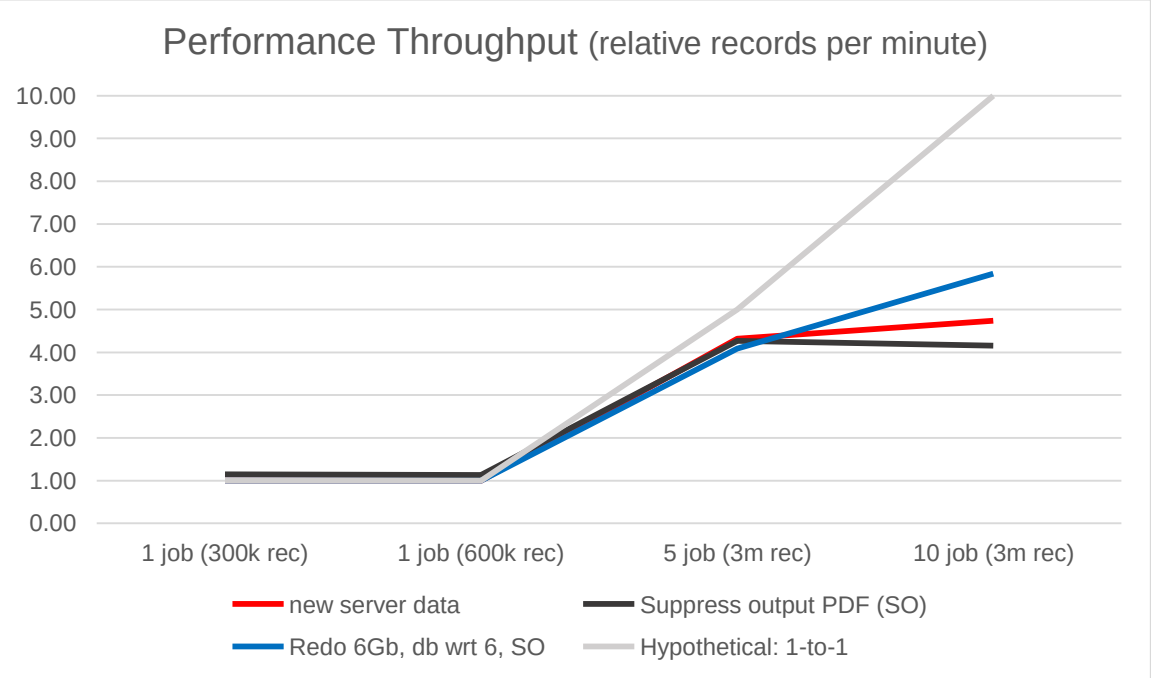


As we continued to run dbconsole with each iteration, we found that, while there is less time spent “in the brown” doing configuration work, we now have a new large area in the “pink” representing “other”. Further drilling down showed this was due to “log file sync” activity.

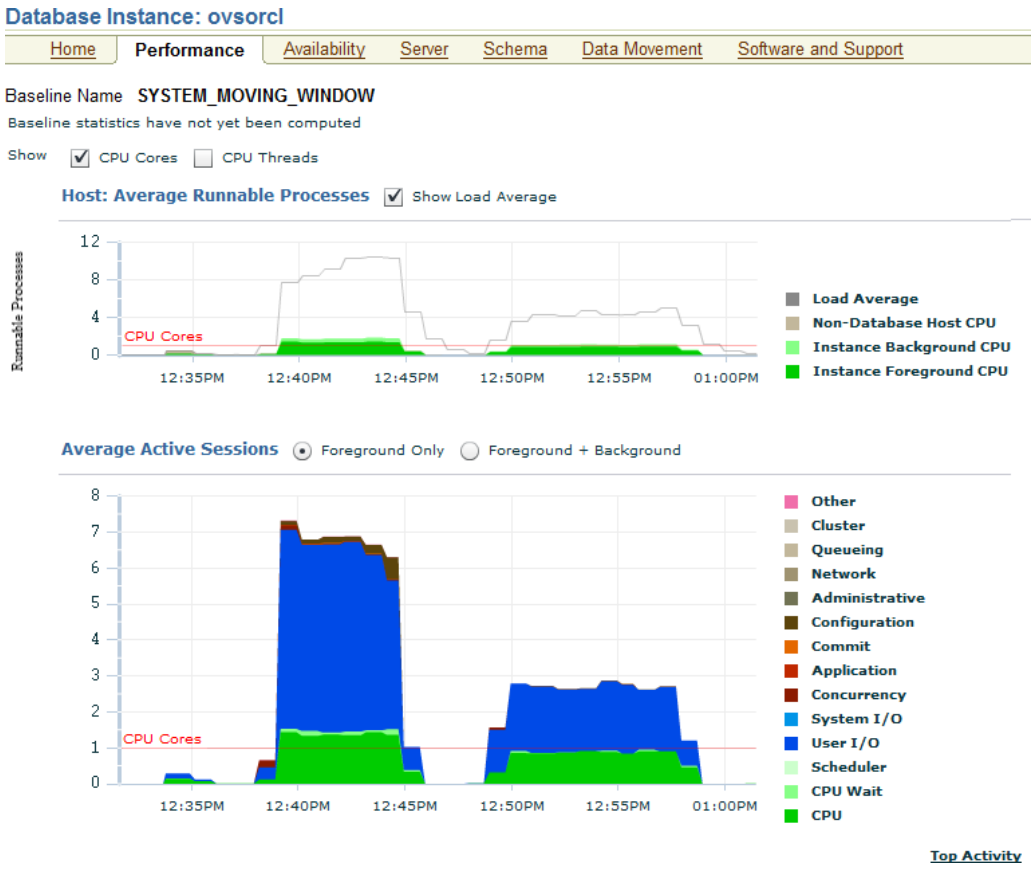


Iteration 4. We noticed in the previous run that all six of our redo logs were still almost always full and waiting was occurring on them. We saw that our jobs were issuing updates faster that the redo logs and writers could keep up. As this database system has plenty of disk, and system analysis showed that the database system CPU utilization was still under 50% of the system maximum, we once again increased the size of the redo logs and number of dbwriters.

Our fourth iteration increased the six redo logs to 1GB each (for a total of 6GB of redo log), and increased the number of dbwriter processes to six.



Looking at the dbconsole, we can finally see that all is well; the 10 job use case #4 is on the left, and the five job use case #3 is on the right:



Iteration 5. At this point, we tried increasing to twelve 1GB redo logs, either serial or as six sets of parallel active redo log groups, but performance became worse again. Operating system analysis showed we had maxed out the throughput of our disk array.

Iteration 6. For iteration 6, we tried increasing from six to eight dbwriter processes, but once again, our performance was slower and the operating system analysis showed we had maxed out the throughput to our disk array.

The reason for no further increase in performance was the same for both iteration 5 and 6:

Our system, setup as a VM, uses a Xen-hypervisor disk group which is a remote mount for all disk storage. Both the redo logs and the actual database are on the same disk group, and at this point, for iteration 5 and 6, we had maxed out the disk group's data throughput.

To get any further performance, we would have to reduce contention on the existing disk group by:

- Putting the redo logs on faster disks (separate fast disk or consider using SSD)
- Putting each of the redo logs on separate disks (multiple separate fast disks or consider multiple SSD)

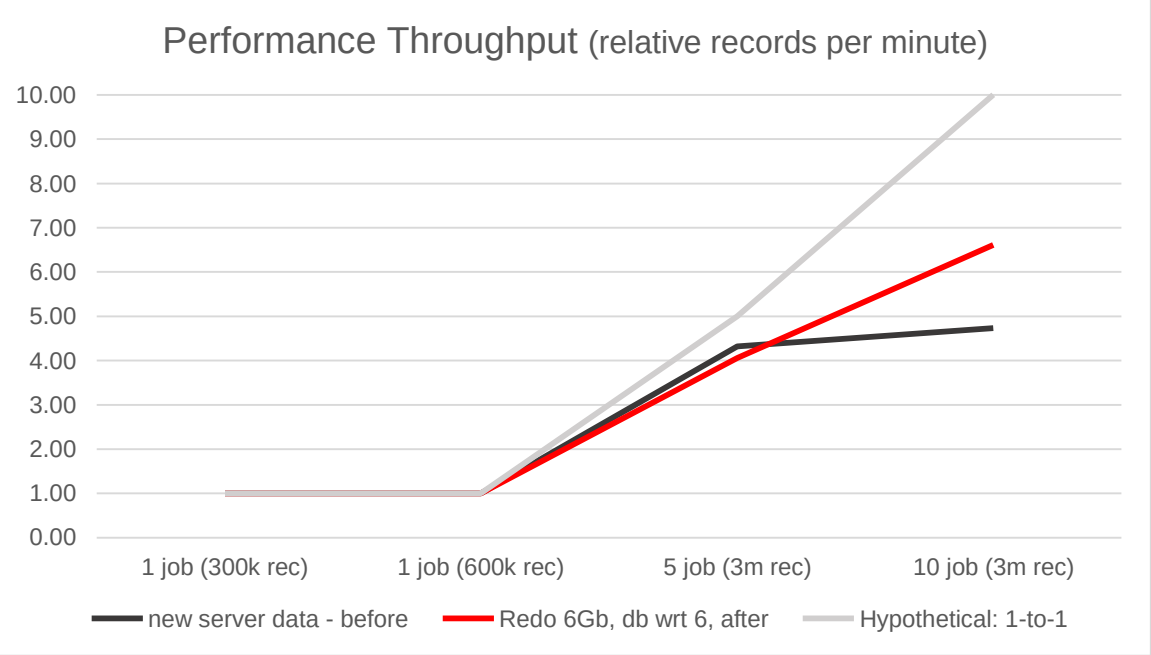
It would be possible to keep increasing our performance, perhaps requiring more dbwriters, simply by doing one or both of the above. The same configuration with faster separate redo log disks should achieve much higher numbers

in the same test. On a system preconfigured with fast flash disk, such as an Oracle Exadata, this sort of tuning should show higher relative throughput than the system shown in this case study.

Our study ended before reaching the point of running with the suggested additional hardware.

Final result

With PDF enabled again, we re-ran to compare our results from before and after tuning our database:



As the red line in the graph shows, the result after performing some database tuning is much closer to the ideal gray line. As we increase batch workload, the resulting throughput scales commensurately.

Conclusion

Batch Jobs in EnterpriseOne can scale by being divided into separate processes provided that the database is tuned to handle the extra load imposed and has the hardware resources required to achieve the performance desired.

Recommendations

- When planning database capacity, the number one thing that can lead to a highly-performing database is having the fastest IO system you can afford, and then properly tuning your database to use it.
- For tuning your database, we recommend the Oracle Database Diagnostic and Tuning packs, as they can quickly reveal issues with how your Oracle is tuned, in conjunction with analysis of system hardware and OS metrics, such as those available in the Oracle Enterprise Manager "OSWatcher" scripts, which make collecting and comparing OS information much simpler. The identification, analysis, and rectification of the database tuning opportunities would have been much more complex and difficult without the Oracle Database Diagnostics and Tuning Packs.

Considerations

- This database was tuned specifically for the load in question on the test system used. For other types of database loads and systems, this exact tuning recipe will most likely not be a so-called "magic bullet". For all performance analysis situations, one must collect operating system, network, and server hardware metrics from all involved servers, as well as performance information that is available from your database. This information will have to be studied to determine how exactly to tune the systems in question for best performance.

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Hardware and Software, Engineered to Work Together