The Case of the 2× Slower Report

Why does a report run twice as fast on an identically configured sandbox? It's not the execution plan. Method R Workbench makes it easy to find out.

by Cary Millsap

Comparing how programs spend their time is a common Method R Workbench use case.

Comparing how a program spends its time on two supposedly identical systems can reveal opportunities for improving performance on the slower system.

Knowing details about individual database and system calls—an Oracle feature that is available only in trace data—can be vital to understanding how a program spends time.

Adding memory may not be the best way to solve buffer cache abuse. It may be faster, less costly, and less risky to optimize or reschedule your workload.

Problem

A report that runs for one hour on the sandbox system takes two hours on the production system. Both systems use identical hardware configurations. A nightly mirror-copy process ensures that both file systems are byte-for-byte identical each morning. All configuration parameters, optimizer statistics, and data are identical, and both reports are executed using the same input arguments. Both systems run the exact same SQL statements and use the exact same execution plans. Why does the production report take twice as long?

Plan

The difference could be anything. There are thousands of ways that your report on one system can spend your time differently than on another system. You'd drive yourself crazy trying to check for them all. We solve the problem by observing how each program spent every microsecond of its time.

Enabling Oracle extended SQL trace for a report execution on each system will create two trace files. Loading these files into the Method R Workbench application will give us a beautifully formatted HTML *profile* showing you how each report has spent its time.

Analysis

As expected, the production profile explains a 2-hr report execution, and the sandbox profile explains a 1-hr execution. But the production profile contains an extra hour of time spent executing `db file sequential read` calls. Why would there be an extra hour of reading on `prod`?

A difference in time spent reading can be caused only by a difference in call latencies, or a difference in call counts. In this case, it is both. The latency per call is about 3× longer on `prod`, which is curious. But there are 22× more calls on `prod`, which is extraordinary. Why would two identical systems running the same report on the same data need to execute such vastly different numbers of read calls?

The profile shows that the call count difference is not due to differences in row counts, or buffer cache touches, or execution plans. All those measurements are the same on both systems. We need to know which database blocks each report is reading.

Method R Workbench can show you this information. For every individual read

call, you can see specifically which blocks were read, how many blocks were read, and how long the call took. Method R Workbench can report the information you need, grouped by Oracle Database block ID.

As everyone expected, both systems read exactly the same blocks from disk into the database buffer cache. The surprise is how many times each block was *re*-read. On sandbox, the report read each block only two or three times in the worst case, but on prod, the report read Oracle database blocks an average of 12× apiece. It read some blocks as many as 70× apiece!

The two systems may look identical, but they have one important difference: prod serves thousands of concurrent programs, while sandbox lopes along mostly idle. On sandbox, when the report needs to visit a block in the buffer cache a second, third, or even seventieth time, the block is usually cached, waiting patiently. But prod is so busy serving other programs that by the time the report needs its second access to the block that it had just read just moments ago, some other program has already aged the block out of the cache, requiring the report to read it again—a second, third, or seventieth time.

Of course, the extra read calls on prod create a second type of performance penalty as well: disk I/O queueing. The increase in read call counts makes it more likely, for every program on the system, that the next read call will have to wait behind some other program's read call, which is responsible for driving the read call latencies up by 3×.

The increases in both call counts and call latencies add up, causing the overall experience duration on prod to double.

Solution

The database buffer cache on prod is too small to handle the load being imposed upon it. Allocating more memory is one option, but it's not the only one you should consider:

1. Is the report written to run efficiently? Maybe with a little attention to the SQL, it won't need to visit the same blocks over and over.
2. Are there other inefficient SQL statements in the prod workload that are ruining the database buffer cache for everyone?
3. Could some buffer-cache–intensive programs be scheduled to run in the middle of the night instead of during peak hours?
4. Would allocating more memory to the database buffer cache help? How much to add requires analysis and experimentation, and you have to beware of side effects. For example, if you use Oracle® Exadata®, then making your buffer cache bigger may degrade the performance of other programs that use Smart Scan.
5. Will you need to upgrade your system? Maybe you've already allocated all the memory that you can.

Technology

Method R Workbench is easy-to-use, high-precision *Oracle time measurement software* for software development, code reviews, performance tests, concept proofs, hardware and software evaluations, upgrades, troubleshooting, and more—for Oracle developers, DBAs, and decision-makers in every phase of the software life cycle. The block ID report described in this monograph is a standard report that is shipped with the product.

◇ Method R™
method-r.com
info@method-r.com