Solving the Unsolvable Performance Problem

How do you solve production performance problems? The method you use may be structurally incapable of helping you find certain types of performance problems. Changing how you look at your system makes all the difference.

by Cary Millsap

The way you look at your system can block you from understanding your performance problems.

Many Oracle technologists don't know how to answer the most fundamental question about performance: "How does a given program spend its time?"

Tracing explains how a program spends its time. This enables you to find wasted time, no matter where it occurs in your technology stack.

Method R Workbench makes it easy to work with trace data, even when you have thousands of trace files.

The ability to measure the response times of your business's more important programs helps you solve more problems, more efficiently.

Problem

We meet people all the time who have suffered a particular performance problem for months, or even years. The people we work with are plenty smart and plenty motivated. Most of them have spent loads of money on hardware upgrades and consulting assistance. How can their problems remain unsolved for so long?

Most technologists view a computer system from the *supply* perspective. They measure the system's internal resources and seek patterns that might imply bad behavior. Unfortunately, many performance problems are invisible to this method. However, the same problems are easy to find when you view the system from the *demand* perspective.

Plan

You can solve more problems and waste less time when you stop looking at a system as a collection of resources and start looking at it as a collection of *user experiences*. For example, imagine that your business tells you that BalanceInquiry is too slow. Then your success will be measured by how much you improve the response time of BalanceInquiry. No other metric matters. Your mission, then, begins with answering one critical question:

> *How does* BalanceInquiry *spend its time?*

The problem is, tools like AWR and Oracle Enterprise Manager don't answer that question. It's even worse than it sounds: those tools can actually weaken your understanding of what your programs are really doing. The performance ideas your tools inspire may actually be the cause of your suffering.

Analysis

You can answer the big question—"How does my program spend its time?"—by representing a program's duration in the same quantity-and-price format you would expect on an invoice or a restaurant receipt. This type of report is called a *profile*.

A profile's bottom line shows the duration that the person who ran the program actually felt, and the durations in the table sum to the bottom line. The format makes it clear that there are only two possible root causes for any performance problem: (a) some call count (quantity) is too high, or (b) some call duration (price) is too high.

Subroutine	Duration (seconds)	Calls	Mean duration per call (seconds)	Max duration per call (seconds)
db file parallel read	15.624	3	5.208	5.407
waiting for CPU	13.844	21	0.659	13.657
CPU: EXEC dbcalls	0.261	8	0.032	0.245
11 others	0.274	4,170	0.000	0.000
Total (14)	30.003	4,202	0.003	13.657

A *profile* represents a program's duration as a table of quantities (calls) and prices (durations). It's an invoice for response time.

Solution

You can't create a profile like this for an Oracle application program using AWR (or even ASH) data. But Oracle does provide the information you need to create a profile with its *extended SQL trace* feature.

Tracing is easy, but it can generate a lot of data. With our Method R Workbench, you can profile individual user experiences even if you have thousands of trace files. The result: a method for solving problems you wouldn't have solved any other way.

Example

A wealth management application has had performance problems for a year. The BalanceInquiry feature is normally almost instantaneous, but several times each day, it consumes 20 s (seconds) or more. The behavior is eroding customer faith in the application.

The application isn't designed for easy tracing, so we trace the whole system for an hour. We use Method R Workbench to find the 20 s BalanceInquiry execution. Its profile reveals that 19.8 s is consumed by a single log file sync call.

Further investigation using information that is available only in trace files reveals that this specific log file sync call was blocked by a log file parallel write call being executed at the same time by the Oracle LGWR process. Other users were similarly blocked by the same write call.

We found this problem within 30 minutes of receiving our first batch of trace files. How could it have evaded detection for a year? It's because the "average" log file sync call isn't a problem. AWR shows that the average log file sync duration on this system is only 0.025 s.

In hindsight, of course you can see in v$event_histogram that log file sync calls do take 19+ s sometimes. But the team didn't act on that information because log file sync was unremarkable—v$event_histogram shows fifty other call types that behave the same way. Even if someone *had* proposed log file sync as BalanceInquiry's problem, the organization probably wouldn't have had the resolve to fix it, because there was no cause-effect link to justify the cost.

…And so the intermittent BalanceInquiry problem persisted for *a year*. Before tracing, there were hundreds of possible root causes, none provable or disprovable with aggregated data. But after tracing, there is no doubt: log file sync (and thus log file parallel write) is the definite cause of the problem.

Method

This is a story we repeat over and over: intractable problem; trace it (trace *everything* if we have to, and sift through tens of thousands of trace files in just a few minutes with Method R Workbench); profile the interesting user experience; now we know where the wasted time goes.

» In an airport management system experiencing application timeouts, trace files revealed a row lock held for ten seconds because of an intermittent disk latency problem. Nothing in the system-wide statistics implicated either locking or disk I/O as a diagnostic priority.

» In a global convenience store's test kitchen where a simple ingredient search took two minutes on a $6,000,000 computer, trace files revealed a badly chosen Oracle parameter value and an application design mistake. Neither problem presented symptoms visible in system-wide statistics.

» In a professional society that had just moved to the cloud, trace files revealed response times dominated by now-longer network latencies. Luckily, a bad but easily fixable coding habit was causing a lot of unnecessary network round trips. Their Oracle monitoring tools, like most, discard network I/O call data.

» In a GPS location monitoring application that intermittently overwhelmed its CPU capacity, trace files revealed a bad coding habit of dynamically generating and parsing thousands of unique SQL statements per hour. Monitoring tools showed the bad practice but undervalued the impact.

» In a finance management system that overran its nightly batch window, trace files revealed the problem to be not an Oracle Database problem at all, but a suboptimally programmed third-party application running between specific pairs of SQL statements. Monitoring tools did report the database as mostly idle but provided insufficient information to help solve the problem.

Everywhere we go, we find problems hiding behind averages. The symptoms vary from one system to the next, but the same method works for all of them: you answer the question, "How does my program spend its time?" The answer begins with tracing.

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 sifting and profiling operations described in this monograph are standard product features.

Method R Workbench 9

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