An Industrial Engineer's Approach to Managing Oracle Databases

Robyn Anderson Sands
System Design Architect
Cisco Systems, Customer Advocacy

Scope

This paper is an introduction to several concepts used in the manufacturing industry to improve efficiency. Some of these concepts will be compared to existing, well known methods in database management to demonstrate commonality between the disciplines. Other concepts will be explored to determine if there is a potential to develop new tools for database management. This paper discusses variation and why variation should be minimized where practical. The potential use of variation as a tool in performance measurement will be explored.

Contents

Introduction

Some of the most effective database management practices have much in common with the disciplines of Industrial and Quality Engineering. In the manufacturing world, products designed with complex processes and inconsistent parts are difficult, and therefore expensive, to produce. Designing for manufacturability results in better quality at a lower cost. Similarly, database systems designed for manageability can be supported with better performance and less effort.

The manufacturing industry has been in pursuit of efficiency for over 100 years, beginning with the concept of interchangeable parts for gun production, moving through assembly lines and the industrial age and later championing lean production and various quality techniques. These adaptations were necessary to create scalable production systems capable of meeting consumer demand.

The information industry is moving through a similar growth process albeit at a much faster pace. Database systems were once similar to handcrafted works of art, sculpted to meet a unique purpose. As technology progresses, systems are proliferating in order to keep up with the demand for information, resulting in a growing number of databases for administrators

to manage. As the quantity of systems continues to increase, the systems themselves are growing astronomically, resulting in the need to manage massive amounts of data while still providing the performance levels required by the business.

In response to these trends, there is a corresponding trend toward the engineering approaches I learned at school and in the early years of my career. For example, Method R, as presented by Cary Millsap and Jeff Holt in "Optimizing Oracle Performance", echoes concepts dating back to Frederick Taylor and "The Principles of Scientific Management". Taylor advocated the use of time studies, whereby each step required to complete a process was measured and analyzed and that which was unnecessary was eliminated. A similar thought process is the focus of 'The Goal' by Eliyahu M. Goldratt; efficiency can only be improved by locating the bottleneck in the process and increasing capacity or reducing the load.

There is still more to be learned from the manufacturing and quality disciplines. The goal of both is the creation of a quality product via an efficient process and what is performance if not efficiency? Efficient code, efficient use of hardware resources, efficient use of human resources, all working toward the goal of providing an efficient tool for the business. If the database system is designed efficiently, it will meet its performance expectations. This does not mean that an efficient logical design will compensate for inadequate hardware nor does it justify the inefficiency of KIWI (Killing It With Iron). Appropriate hardware is a key design component and over purchasing hardware is simply another type of inefficiency. A lean approach requires careful analysis of requirements and a resulting design that is prepared to scale to meet future needs.

This brings us to the enemy of efficiency - a nemesis called variation. Quality engineering seeks to drive out variation by using consistent processes and tools. Components and sub-assemblies are built to increasingly tight tolerances because it is recognized that by controlling variation at the detail levels of part production, the end result is also more consistent. Quality control focuses on measuring and then discarding products that do not meet requirements while quality engineering targets reducing variation in the process to minimize loss from scrap.

I propose that the elimination of unnecessary variation is one of the key tasks of creating an efficient database system, and the measurement of variation in response time can assist in the identification of potential performance issues. But let's begin by looking at some well known database management methods to see where they are similar to the techniques that led to revolutionary changes in the manufacturing world.

Component Interchangeability

The use of interchangeable parts was one of the first advances made in manufacturing, and the munitions industry was first to take advantage of this approach. Before the advent of interchangeable parts, a broken gun

either went back to the gunsmith for repairs or it was useless. This could be inconvenient, especially if you were in need of your weapon at that particular moment. The concept of using interchangeable components was originated by General Jean-Baptiste De Gribeauval in 1765 and later pioneered by Honoré Blanc, a French gunsmith working for Napoleon. Interchangeable parts allowed guns to be assembled quickly and reliably, and as a side benefit, provided the option of spare parts for repairs.

The Optimal Flexible Architecture (OFA) standard was the first documented approach to Oracle system management I read that reflected an industrial engineering mind set. This standard describes an approach to database file system organization that results in something comparable to a manufacturing bill of material, or BOM. By clearly segmenting the key components of an Oracle installation, each component can be viewed as an individual sub-assembly and is therefore able to provide a function independent of other components. These sub-assemblies are also reusable as needed. For example, if the Oracle executables are completely separated from the database files and log files, the directory containing the executables can be replicated or mounted to another system. When it's necessary to patch or upgrade a system, a new mount point can be installed without impacting the existing system until it is confirmed valid and usable. By maintaining actual database files in a separate, uniquely identified path, they too can be restored, replicated or mounted to a new server. With careful planning and a documented approach, components of a database system can be assembled and reassembled to minimize the impact of system maintenance on the business.

With the advent of Real Application Clusters (RAC), Automatic Storage Management (ASM) and the current trend towards consolidation, the likelihood of multiple $ORACLE_HOME's on a server has increased. By designing a system to meet the goals of OFA, management of increasingly complex environments is simplified. Unfortunately, many debates of the current relevancy of OFA degenerate into discussions of naming conventions. In reality, the names of mount points and files systems are not the focus of OFA. What is key is the separation of components based on their functionality and the ability to cleanly alternate between multiple environments. Also key is the documentation of the standard that is implemented, and that it is consistent across all environments. Implementing a standard will do little to improve efficiency in administrator performance if a different standard is set for each server.

As an additional benefit, careful use of environment variables within scripts and tools results in packages of database management scripts that can be installed and reused consistently. Hence, all database management scripts become another 'interchangeable part', easily installed and functional with minimal troubleshooting required. If all environment variables are defined in one file and all servers are configured consistently, a script that makes good use of environment variables will function without changes on all database servers. If all user id's are set

to call a single script that sets their environment based on the instance they need to access, then all changes can be made in a single location. This not only makes changes easier, it decreases the likelihood of processes that fail post upgrade because someone forgot one of the files that needed to be updated to the reflect the latest version.

This section has made frequent reference to 'the goals' of OFA. Rather than repeat what is already well documented, I refer the reader to the original OFA document, available at Hotsos.com. Begin at the end of the paper. Read the requirements section, read the list of recommendations and then decide if your current configuration meets the goals described. Then proceed to the beginning of the document to gain a full understanding of the potential value. It is not necessary to follow the standard to the letter but it is highly advantageous to ensure that your systems are able to meet the requirements and that all systems within your jurisdiction are consistent. If you must deviate from the recognized standard, document your changes for those who will work on the systems you implement in the years to come.

Time Studies

In 1911, Frederick Taylor published the 'The Principles of Scientific Management' and the image of the Industrial Engineer with his stopwatch in hand was born. Taylor's assertion was that by analyzing each movement of a worker, the most efficient series of movements could be found and the unnecessary movements eliminated, therefore reducing the total amount of time to produce a product. Taylor had many detractors; his methods were seen as dehumanizing and in truth, laid the groundwork for much of the work to be done by machines in the future. However, his conclusions were accurate; the elimination of the unnecessary is the fastest way to speed up a process. Or as it has been noted in multiple presentations at various Oracle conferences around the world, the fastest way to do something is not to do it all.

Taylor's scientific management process was also sound. A specific process needed to be defined, measured, redesigned and remeasured until the optimal approach was found. Similarly, in Method R, the same series of steps is followed until a database process has achieved a satisfactory response time. Key to Method R is identifying the processes that are most critical to the business function and to repeat measurements at each change point until success is confirmed.

Oracle's wait interface, while not perfect, is recognized as one of the key advantages to the Oracle RDBMS. No other database is as highly instrumented. As more becomes known about the internal workings of the interface, more tools are available to exploit it and therefore measure actual progress in performance tuning. As instrumentation is added to some operating systems, additional layers of mystery are removed. Similarly to man's understanding of fire, what was once seen as 'black magic' is now predictable and what is not yet predicable is recognized as not yet fully understood.

Key to the effective use of Method R is to approach a tuning exercise as Frederick Taylor might have; define exactly which part of the process is under evaluation and measure its performance prior to making any adjustments. Evaluate the results and determine where time is being spent, possibly 'wasted'. If there is a bottleneck in the process, improving performance anywhere but the bottleneck will be pointless and possibly even detrimental. Read 'The Goal' if you have any doubts about this, or talk to someone who has seen the futility of adding CPU's to a system short on memory or I/O capacity. Once the problem has been clearly identified, work to add capacity to the bottleneck or reduce the processes need for those resources. Measure again and evaluate your progress. Repeat as needed.

Frederick Taylor would be proud.

Value Engineering

Value Engineering (VE) is a systematic problem solving approach that makes use of function-based techniques developed by Lawrence D. Miles in 1947. Value is defined as the ratio of function to cost, therefore value can only be increased by adding functionality or reducing cost. Every Federal agency is required to maintain a value engineering program and the Department of Defense uses value engineering to measure an aircraft program's performance, reliability, quality, safety, and life cycle costs. VE involves planning for the lifetime of the program and ensuring that a maximum value is receive for minimum cost.

While this is an older method that is used less frequently in industry in recent years, the goal of a system that provides a maximum value at a low cost for its complete life cycle needs to be remembered. Often, new software implementations have big budgets at the beginning of a project but funds for long term maintenance are scarce. Systems that are implemented without planning for manageability can become drains on department budgets, creating less value due to unexpectedly high costs.

So what can we learn from value engineering? Plan ahead. Think about the total life cycle of a new system before implementation, including ongoing support. Take the time to configure the system to minimize maintenance tasks, which leads to lower cost overall and therefore, to a higher value. This concept works for database administrators as well; you can only increase your value to an organization in one of two ways; reduce the price an organization must pay for your services or add to your functionality via education and new skills. The educational route is far more rewarding.

Root Cause Analysis

Root Cause Analysis (RCA) is the process of evaluating a series of symptoms until the origination of the problem is found, and then fixing the underlying cause of the problem to prevent a reoccurrence. Sounds obvious, doesn't it? However, when a system support person is awakened at three a.m. to address a problem, the common reaction is to fix the immediate symptom that is causing the pager to sound or the data center to call. By the

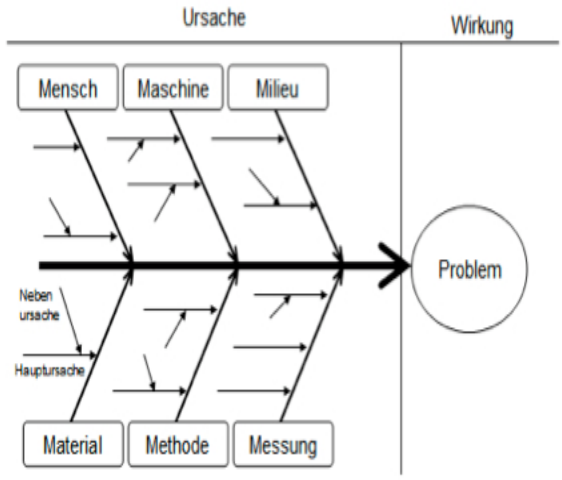
next morning, everything is quiet and the problem is assumed to be resolved, until the next alert.
While RCA is one of the simpler techniques, there is a defined process that makes use of several techniques to assist in determining root cause. The steps to complete an RCA are:

1. Define the symptoms
2. Gather data
3. Identify contributing factors
4. Identify the probable root causes
5. Develop solution recommendations
6. Implement the recommendations, preferably one at a time
7. Measure the results

One of the easier techniques in RCA is the "Five Whys". This approach states that by asking "Why?" approximately 5 times, you will be deep enough in the problem to locate the root cause. The number 5 is a rule of thumb; you may get there in 4 rounds or it may take 7 or 8. In any case, asking questions like a three year old leads you to the answer and the point is to continue asking until you have moved beyond the symptoms.

Various charting methods can also be used to assist in the process of identifying the root cause and eventually, the proposed solutions. Two of the more common charts are the Ishikawa diagram and the Pareto chart. The Ishikawa diagram (figure 1) presents an issue in a diagram similar to the pattern of fish bones, hence it is also known as a fish bone diagram. The contributing factors and visible symptoms are listed along the small bones of the fish, in sections representing major categories. The original categories were Methods, Machines, Materials and Manpower, but these can be adjusted to suit the issue being discussed. All potential factors under each category are cataloged until the root cause is identified.

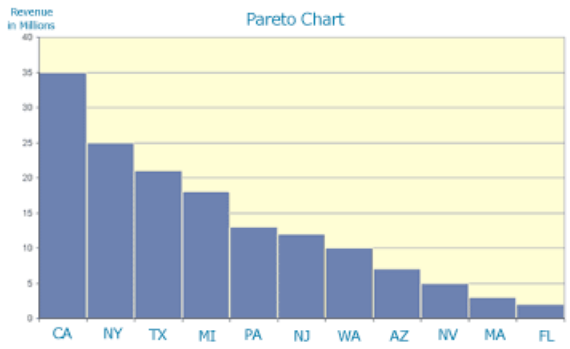
Ishikawa Diagram Figure 1



The Pareto method charts the occurrence of multiple problems and seeks the 20% of the work that will resolve 80% of the problems. This is known as the Pareto principle: the majority of the problems are caused by a few key issues. The need for the actual charts depends on the complexity of the problem being discussed; for many issues, the process of asking questions until the root of the problem is found is sufficient.

There are several variants on the Pareto chart (figure 2) but the most common version is a bar chart used to track the frequency of groups of occurrences in descending order of magnitude. The cumulative values may be plotted as a line moving left to right above the frequency bars.

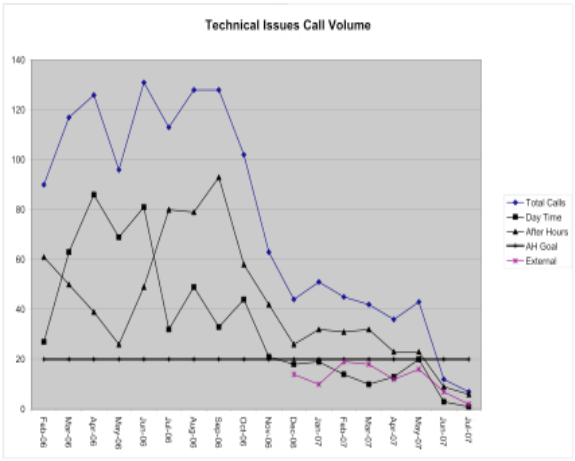
Pareto Chart Figure 2



In an operations center similar to thousands of others, the number of after-hours calls for database administrators averaged about 120 calls per month over a six month period. (Call rotation was one primary DBA and one backup DBA for one week at a time. Rotations were divided between 3 administrators so each had one week as primary, one week as backup and one week off.) This did not include automated pages from systems; only actual phone calls from the data center outside of normal working hours. Part of the growing sense of frustration stemmed from the fact that many of the calls were preventable, but it was also recognized that focus on project work was keeping administrators too busy to handle routine maintenance work proactively.

So a new project was initiated, one that provided a budget for charging time to address the root cause of any issue that resulted in after-hours emergencies. If an administrator wanted to use this project budget to fund necessary work, there was only one rule: the proposed change had to stop the problem from occurring, not just apply a temporary fix or alert someone after an issue had occurred. Of course, alerts and quick fixes are sometimes necessary to keep systems available to the business, but these steps were already being taken within the existing troubleshooting and maintenance efforts. What was missing was the follow up to eliminate the source of the problem. As the Pareto Principle describes, a quick analysis of the reasons for the majority of the calls led to the identification of a handful of fixes. Once this first round of root causes were addressed, the number of calls began to drop rapidly. Once results were demonstrated, there was increasing support for the project and additional root causes were identified and repaired. Three months after project inception, the number of after-hours calls in one month had dropped to 43. Over the next six months, the number of after-hours calls continued to drop until they remained below the targeted 20 call per month maximum, where they remained through the end of the year when the project was closed.

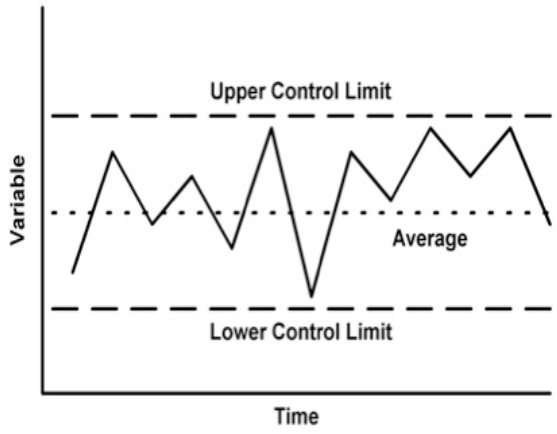
Call Volume Chart Figure 3



Statistical Process Control

Statistical Process Control (SPC) involves identifying the range of acceptable values for a process and monitoring for variation as an indicator of current or future problems. Variation comes from two sources; there is controlled variation that represents a natural fluctuation in a process and then there is uncontrolled variation, or special causes. Uncontrolled variation can be an increase in variance, or it can show that a process is trending in one direction and moving off target. SPC was originated by Dr. Walter A. Shewhart at Western Electric Company in the 1920's, as was the control chart (figure 4) which displays individual outcomes between an upper and a lower control limit.

Control Chart Figure 4



www.tangram.co.uk/GI-InfoSheet06-ProblemSolving.html

The control limits are set to indicate the boundaries of normal, or controlled, variation while any data points that fall outside these two limits are uncontrolled variation, or special causes, that merit investigation. When using SPC, a process is measured and methods are identified to minimize variation. The control limits are tightened after each improvement, until they represent an acceptable range. The process continues to be measured and monitored for any sign that variation is increasing. If the process is getting out of control, the goal is to correct the problems before parts are produced outside of the specified tolerances, reducing the incidence of scrap.

This tool is a natural for monitoring database processes, with a few adjustments, and many administrators use some form of a control limit monitoring even if it's not a formal process. Upper control limits should be set based on process capability and business need. Lower control limits may not be needed, as it is rare that anyone would complain if a process were too fast, however a 'too fast' completion can indicate a failure - perhaps data was excluded that shouldn't have been. One key difference in control charting for system processing is that upward trending can be a type of controlled variation if it is in proportion to growth in the data set, illustrating the importance of defining 'what is normal' for the system before setting the control limits.

The point of this process is to define normal expectations for the system; where the normal variation lies and where there is uncontrolled variation that can be removed to make the process more consistent. Not all database processes need to be blazing fast; consistency is an important factor to performance, and user satisfaction will

be higher when processing times meet their expectations.

Design of Experiments

Design of Experiments (DoE) is the process of planning and executing an experiment; the evaluation of process, the collection of data and the evaluation of results by statistical means. The goal is to identify relationships between the variables, develop models and predict system behavior in the presence of both controlled and uncontrolled variation.

Although he is considered controversial in some statistical circles, the methods defined by Dr. Genichi Taguchi are thought provoking. Taguchi contended that conventional sampling methods were incomplete as it is not possible to collect a random sample of future conditions. Traditional approaches sought to remove variation from the experiment. Taguchi made variation the focus of his experiments, and the goal became the determination of a process capable of performing consistently in spite of variation. The Taguchi method makes use of loss functions, orthogonal arrays and a concept called 'signal to noise ratio': a method for comparing the average to the variance. A complete series of Taguchi style tests for a large database system would likely be prohibitively expensive. However, gaining an understanding of Taguchi's concepts and goals provides an interesting perspective on performance issues.

A common example used to explain this concept in the manufacturing industry is a paint shop in Arizona. Traditional testing approaches require a tightly controlled environment for testing the paint, the application process and the results; Taguchi would argue that variation in temperature was inevitable and that a variable temperature should be part of the testing process. If a production shop were built based on the results from the traditional testing methods, maintaining the required level of control to produce the desired results could become a time consuming and expensive operation. Choosing the paint product and process that are able to perform consistently in the presence of temperature variation results in a process that is less expensive and more reliable.

What if we sought out the potential variables a database system is likely to encounter and designed the system to perform consistently in spite of the variations? The greatest variable an online transaction processing (OLTP) system faces is the user while an online analytical processing (OLAP) system will face most variations within the data itself. Some systems may have resource limits that create potential for variance in hardware performance. Recognizing potential variants and making design choices to minimize their impact can result in a better performing, lower maintenance system.

Variation is the Enemy!

What if the performance of a process were measured not by just its response time, but also by how much response time varied? Although it will always be crucial to understand and focus on the key processes defined by the business, those key processes can be affected by the variation in another query or update

running in the same timeframe. While the business users may be able to define the critical processes, they may not recognize all the interrelationships in the processing schedules and the potential resource impact individual jobs can have on each other.

A recent tuning exercise brought this concept into clear focus. A daily sales and bookings (DSB) report tended to slow considerably toward the end of any quarter and especially at the end of the fiscal year, just when the report was most critical to business users as they worked toward meeting their targeted sales goals. Analysis of the wait times did not show problems with latches or locks until the report had failed several times and the impatient users had resubmitted multiple requests for the same data. The DSB report had been created years earlier and had been modified many times, resulting in layers of views and interrelationships with other reports that made changes to the code a scary proposition. Prior tuning attempts had helped temporarily but the results never lasted very long. In the fourth quarter of FY07 it was decided that the job needed to be thoroughly investigated before the process hit critical mass at year end.

Monitoring individual user executions showed that response times were highly varied. At best, the report finished in about 12 minutes but sometimes it failed after 20 minutes due to a timer on the report server. When the code was traced for a single execution, the predominate waits were read waits. The data was also highly skewed as the sales numbers had trended upward over the years. In order to minimize the impact of the skewed data, the central table was partitioned by fiscal year and sub-partitioned by month. Partition pruning minimized the reads and the next round of trace files showed that the majority of the response time was now spent in CPU waits. This led to the discovery of a clever little function buried deep within a view that was being used to determine the months that should be included in a specific quarter. Unfortunately, this function clearly didn't pass the scalability test. Early in the quarter, the function performed well but as the number of records to be processed increased, the function created a CPU bottleneck and the more users that tried to retrieve the data, the more work backed up behind it.

The solution was simple; the function was eliminated and two columns were added to the calendar dimension to provide the data. After this change, CPU usage became negligible under any data load. The report was faster but more importantly, it became more consistent. Response time predictably increased toward the end of the quarter, but execution times remained between 6 to 11 minutes. Not all variation could be removed from the process; users can still submit multiple requests and data will always be skewed due the natural fluctuations in sales but the new queries are written to minimize the impact of these known variables.

Variation as a Tool

Having seen the potential for variance to identify older code that no longer functioned well under current data loads, an analysis of the data warehouse job run times was needed to see if variability could be used to identify

other code issues. The data warehouse included a table of job completion times which was used to monitor performance. The goal of this experiment was to compare average response times to variance, using an approach inspired by Taguchi's signal to noise ratio concept.

Collecting the average, median, standard deviation and variance for all jobs that had at least 12 executions over fiscal year 2007 showed that some processes had enormous amounts of variance in their processing times. The variance to mean ratio is a measure of the degree of randomness in the distribution of a given phenomenon. When this calculation was applied to the job execution data, it resulted in a sort of 'predictability factor': the lower the degree of randomness, the more consistently a job performed. The results were logically consistent with job performance history; jobs with the highest variance to mean ratio were some of the biggest troublemakers in the job schedules. The data warehouse team concurred that the identified jobs held the most potential for tuning within the code but pointed out some notable absences from the top twenty. In the past, the team usually focused on the long running jobs as the performance issues. Under this new approach, a job with a long execution time and a lower level of variance indicated that a large amount of data was being processed in a reasonably efficient manner.

One component of statistical analysis involves removing data points that are related to special causes. Several different approaches to identify and mark special causes of variation were tested and a few of them look promising for future use. Especially when the data will be used to set future control limits, it is critical to remove any data points that result from an abnormality outside of the actual code processing. For example, say one night the archive log mount point filled up due to a failure to migrate the archive log files to the backup target and this caused a job process to hang. The data point for that specific job execution during the service interruption is not representative of normal processing, so it should be removed from the statistical analysis. (Of course, the problem that resulted in the full file system should be evaluated to determine how to prevent a reoccurrence but that is separate from the data analysis.) This is especially important when evaluating variance over a shorter period of time with fewer data points.

For this experiment, the raw data was sufficient to test for a relationship between high levels of randomness and performance issues. Options for identifying and marking special causes data points falls outside of the scope of the planned experiment and this paper, but will be presented at a later date.

Sorting the results on the variance to mean ratio, or randomness, in descending order, brought the worst offenders to the top of the list:

20 Least Predictable Jobs (time in seconds)

JOB_NAME	SAMPLE_SIZE	AVERAGE	MEDIAN	DEVIATION	VARIANCE	VARIANCE_TO_MEAN
ba736	15	20291.467	18266	7293.171	53190342.267	2621.316
ba244	24	14720.417	13235.5	5920.793	35055786.428	2381.440
ba654	327	220.933	53	660.884	436767.922	1976.927
ba810	314	5649.035	4559	3044.318	9267870.884	1640.611
ba685	49	26617.163	27231	6228.938	38799664.473	1457.693
ba371	325	1339.065	1069	1381.680	1909040.542	1425.652
ba236	324	3883.188	2740.5	2330.879	5432997.745	1399.107
ba297	23	1823.913	1117	1569.664	2463844.174	1350.856
ba546	388	5413.353	4882	2540.904	6456195.516	1192.643
ba510	12	10791.500	10759.5	3349.294	11217768.818	1039.500
ba483	323	2289.582	1931	1398.884	1956876.909	854.687
ba62	346	493.772	307	605.533	366670.206	742.591
ba457	326	881.810	722.5	790.901	625523.859	709.364
ba706	14	1640.143	940	1046.019	1094154.747	667.109
ba426	12	2025.000	1465	1111.136	1234623.455	609.691
ba298	23	592.043	355	595.571	354704.225	599.119
ba558	151	212.252	1	356.245	126910.390	597.924
wp001	261	280.092	127	398.075	158463.476	565.755
ba51	337	160.288	112	297.527	88522.253	552.271
ba272	866	202.936	147	318.676	101554.293	500.424

At the opposite end of the spectrum were the jobs with very low levels of variance. These jobs had the lowest degrees of randomness and therefore, the highest levels of predictable performance.

20 Most Predictable Jobs (time in seconds)

JOB_NAME	SAMPLE_SIZE	AVERAGE	MEDIAN	DEVIATION	VARIANCE	VARIANCE_TO_MEAN
ba524	12	1.000	1	.000	.000	.000
ba713	180	1.028	1	.165	.027	.026
ba711	177	1.028	1	.166	.028	.027
ba295	239	1.046	1	.229	.052	.050
ba683	50	1.060	1	.240	.058	.054
ba413	241	1.062	1	.259	.067	.063
ba156	245	1.053	1	.274	.075	.071
ba836	147	1.041	1	.284	.081	.077
ba157	236	1.068	1	.298	.089	.083
wp003	207	1.077	1	.302	.091	.085
ba802	145	1.062	1	.317	.100	.094
ba158	235	1.085	1	.322	.104	.096
ba839	116	1.207	1	.407	.166	.137
ba499	47	1.213	1	.414	.171	.141
ba575	330	6.394	6	1.036	1.072	.168
ba254	312	1.263	1	.476	.227	.179
ba650	55	1.291	1	.497	.247	.191
ba556	12	17.750	17	1.865	3.477	.196
ba153	250	1.100	1	.476	.227	.206
ba642	137	1.073	1	.480	.230	.214

The remaining 552 jobs fell at various levels between the between these two extremes. Only jobs with more than 12 executions in the fiscal year were included in this analysis.

Long Live Variation!

So if variation is the enemy, why 'Long Live Variation'? Variation can be minimized but never completely eliminated because it naturally exists in any system. However, we can turn variation into a tool that will help to identify processes with the greatest potential for improvement. If a query has an average completion time of 221 seconds, a median completion time of 53 seconds and a standard deviation of 661 seconds over 327 executions, it's time to start asking why. The median tells us that half of the executions took 53 seconds or less. Why did some of the other 166 executions take so much longer than average? How long was the longest running execution and what caused it? What can be done to make the process more consistent? Is it possible to reduce the variance and bring the mean closer to the 53 second median? Ask your 5 Why's and get to the root cause.

A process with a high level of variance is capable of better performance. It also has the potential to wreak havoc on your database. When a process that executes on average in 15 minutes randomly runs much longer, it requires resources beyond the expected window and creates the potential for conflicts with other processes. Using the Pareto principal to express the concept, high variance processes are likely to be the 20% of the work that will bring 80% of the improvement. Notice the use of the word likely in the last sentence; your mileage may vary and success depends on your knowledge of your database. Statistical information can point you to problems that need your attention, but they are meaningless if you don't understand their relevance to particular database and they can be dangerous when misused.

Proceed with Caution…

The first step in taking an industrial engineering approach to database management is to measure your system under normal conditions. Once you have established what is 'normal' for your system, you can then start the process of monitoring for variation and working to eliminate the uncontrolled variation. It is important to recognize that not all variation is bad; for example, different database instances can require different parameters, user permissions and storage settings based on intended usage. Attempting to remove this type of variation would be counter productive.

Look for the root causes of failure or signs of unpredictability, and address the core issues when possible. If you have a repository of process completion times, mine it for data on process variance and investigate the processes with the highest level of variance in relationship to the average or median completion times. If you find a process that appears to have an excessive levels of variance in relationship to the mean completion time, begin by asking questions to determine what caused the longest executions for that process. It's possible that the variance was caused by an external factor. If the process itself appears to be the source of the variation, capture a 10046 trace and use Method R to determine if the variation can be minimized. Record the results and use them as further documentation of what is normal.

Statistics are tricky; they can help you better understand your system or they can mislead you into believing something that is false. While a comparison of variance to average completion times can be a useful measurement to gain insight into your system performance, it is not intended to be another artificially important ratio. There is no predetermined variance level that guarantees performance. Instead, this metric is intended to be an indicator that can lead an administrator to processes that need attention. Any other interpretation would be a misuse of this information.

Author Introduction

Robyn Anderson Sands is a System Design Architect for Cisco Systems, Customer Advocacy. She worked for over 9 years at Lockheed Martin in Industrial Engineering/ Manufacturing Control, Production Development and Quality Engineering. She earned her certification in Production and Inventory Management by American Production and Inventory Control Society in 1992 and a Masters of Science in Engineering Technology, Quality Assurance in 1995. She has been working with the Oracle RDBMS since 1996.

Acknowledgements

Thank you to Cary Millsap, Carel-Jan Engel, Graham Wood and Jonathan Lewis for answering my many questions and for inspiring the thought process that led me back to my roots.

Thank you for Dr. David Stimley for his assistance with statistical terminology.

References

American Society for Quality website, www.asq.org

DOD Value Engineering Program, ve.ida.org/ve/ve.html

Design of Experiments, Robyn L. Anderson, 1992.

Military Enterprise and Technological Change: Perspectives on the American Experience, edited by Merritt Roe Smith, The Massachusetts Institute of Technology, 1985.

Optimizing Oracle Performance, Cary Millsap with Jeff Holt, O'Reilly, 2003.

Optimal Flexible Architecture, Cary Millsap, Hotsos Enterprises, 1995. www.Hotos.com

The Goal: A Process of Ongoing Improvement, Eliyahu M. Goldratt and Jeff Cox, North River Press, 1992.

The Principles of Scientific Management, Frederick W. Taylor, Harper & Brothers, 1911.

The Toyota Way: 14 Management Principals from the World's Greatest Manufacturer, Jeffery K. Liker, McGraw-Hill, 2004.

Total Quality Management: Text, Cases and Readings, Joel E. Ross, CRC Press LLC, 1999.