IDERA® | SQL Server Whitepaper

USER-CENTRIC TUNING

BY SCOTT STONE

A USER-CENTRIC, COLLABORATIVE, HOLISTIC PHILOSOPHY OF TUNING THE APPLICATION STACK

WHY USER-CENTRIC?

Every application, every delivery system, every piece of intermediate hardware and software is ultimately going to be judged by how well it meets the needs of the end-users. This revelation should not surprise anyone. However, it is surprising how many times individual components are tuned and optimized with the assumption that the sum of optimized parts yields a streamlined whole.

It is just easier for teams to only worry about components that they control. Also, it is just easier for groups to ensure that they have a solid case that any problem must be outside of their area. But, that approach rarely works, because dissatisfied end-users blame everybody associated without respect to relative guilt. When issues impact the business, the consequences also frequently are felt by all as well.

WHY COLLABORATIVE AND HOLISTIC?

All elements of the application stack are inextricably related to each other in how well they serve the end-users together. And the optimum design for each component considered independently can be counter-productive at worst and sub-optimal at best. Consequently, a holistic approach to all related parts of the application stack is necessary.

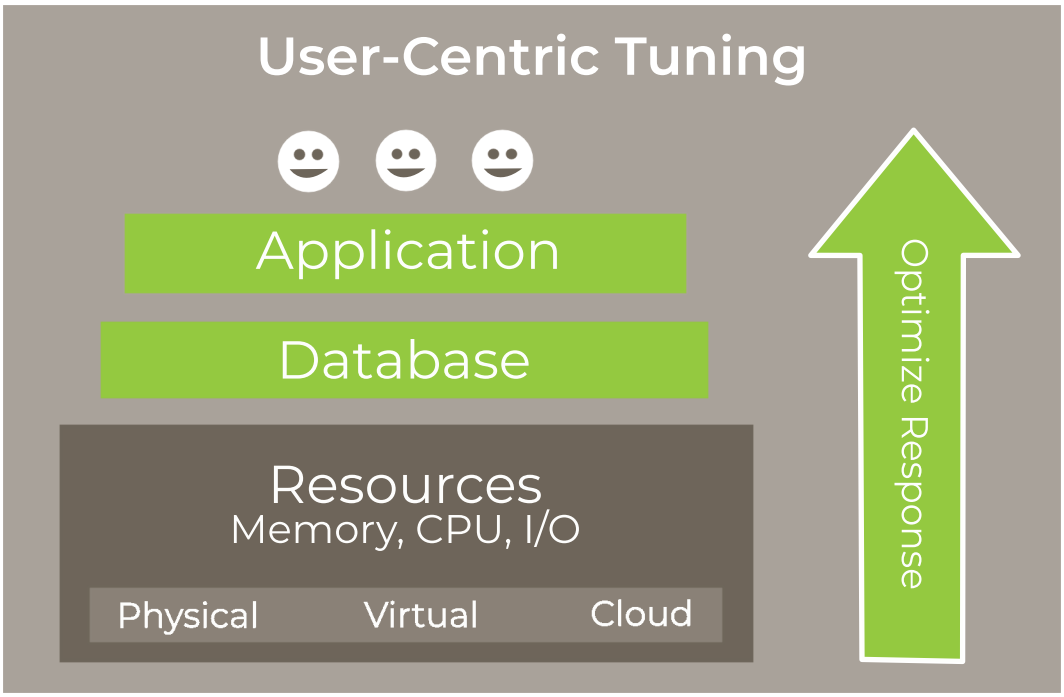
Since different teams and organizations usually control and manage various components of the application stack; a collaborative approach is necessary to achieve the optimization of the whole system rather than the parts.

REACTIVE VERSUS PROACTIVE

These words have been used so much in system management that they became almost meaningless. Proactive is good and Reactive is bad; so we obviously need to be more proactive, right?. However, being efficient when we are required to be reactive is also critical as all contingencies cannot be eliminated.

Reactive tasks are more individual than proactive tasks. Reactive tasks require an efficient diagnosis of a problem, isolation to a root cause component, and quick correction of that root cause issue. We frequently refer to being prepared with systems that increase the efficiency and accuracy of this deductive process as being proactive in itself. From the perspectives of end-users, detecting and fixing problems before they impact end-users is being proactive.

However, our subject here is tuning of the application stack as a whole. Also, that requires a proactive approach. Proactive is methodical and consistent.



User-centric tuning involves optimizing the collective response of applications, databases, and resources (such as memory, CPU, and input/output) for physical, virtual, and cloud environments from the perspective of the end user.

THE APPLICATION STACK

For this discussion, we are considering specifically database applications with structured query language interfaces from the applications to the database. End-users interact directly with applications which deliver some service or benefit to them. The applications and databases may run on resources in physical data centers. They may also run on resources in virtual environments within such data centers. Or, they may run on cloud resources hosted in variable configurations. But in all cases they are consuming physical resources “somewhere” of memory, CPU, and input/output bandwidth.

The teams responsible for these different layers of the application stack vary by organization. Often databases and applications are handled by different groups if not completely separate organizations. And the resources hosting the application and the database portions may differ in hybrid cloud environments. Such cloud environments may host portions locally and other parts in the cloud. Such cloud environments may host portions in different cloud environments or partially virtual environments.

INTERDEPENDENCY

As referenced earlier, the interdependency of the application stack means that the optimization of components in isolation never leads to optimal overall performance. Tuning and optimizing a database for poor performing application will necessarily be incorrectly configured when those same application issues are subsequently resolved. Andhow can anyone possibly size the environments for such changing yet inefficient applications and databases?

BLAMESTORMING

Most performance management and tuning products long made the same claims to “eliminate the blame game”. However, the real effect at best was to optimize the blame game. The objectives are to identify root causes of problems and provide evidence when the particular component should be held blameless.

Winning the blame game may be better than losing the blame game. However, the ideal solution for all would be to truly collaborate in tuning efforts so that all have some confidence. If all teams work together in a unified front, then all can be reasonably certain the resulting performance is the best that is achievable.

A FEW PERFORMANCE MYTHS

Before discussing any optimization and tuning approach, it is essential to debunk a few myths often used to rationalize to not worry about performance tuning.

- Memory is cheap.
- Storage is cheap.
- Processing power is cheap.

People spout these myths when someone wants to throw brute power at an application performance problem. Why waste effort optimizing resource usage when resources are not a problem?

Relatively speaking to past prices, all these statements may be right. But anyone who has had to defend a capital and operations budget knows that getting funding is a different matter. As resources have gotten cheaper, the complexity of applications increased so that we still have the same optimization issues.

The mantra of Information Technology for the past few decades was to “Do more with less” rather than exploiting ‘cheap’ resources with overpowered environments.

Tuning is needed, still.

OPTIMIZE BY LAYERS

Beyond encouraging collaboration across teams, the unifying theme of this approach is to optimize by layers through the application stack.

The rationale behind this approach is to optimize a single layer at a time. However, it is essential to optimize each layer in an order that ensures that the optimization of other layers does not reverse earlier gains.

The disconnected approach of optimizing independently by component can easily lead to an unpleasant game of Information Technology (IT) whack-a-mole.

Database teams optimize performance against a poorly designed application with poorly tuned SQL statements. Then the physical resources are inadequate. Then the application is improved in a way that overwhelms the database optimized to the previous poorly performing application.

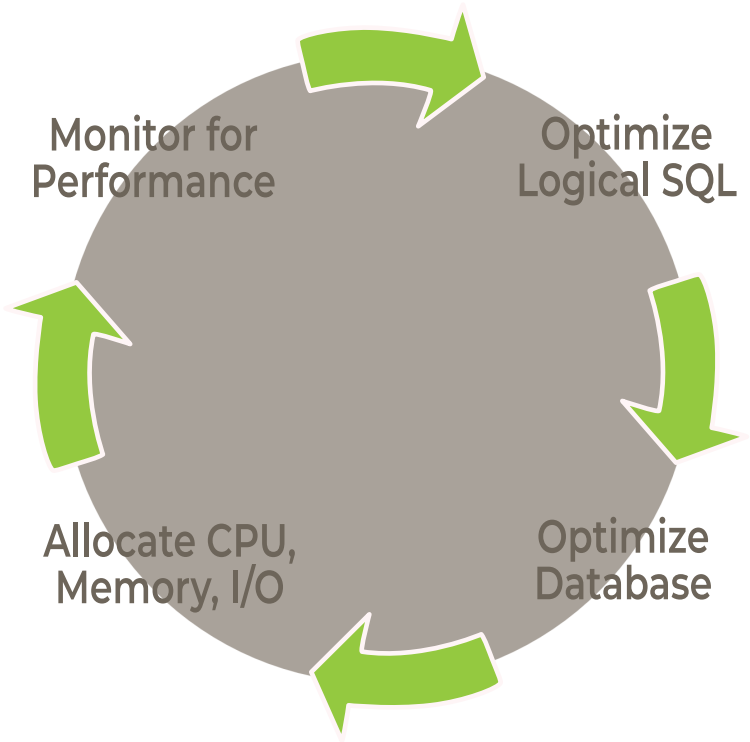
As in any other diagnostic scenario, change a single thing at a time so that you know which changes lead to improvement.

APPLICATION SQL COMES FIRST

SQL from the application represents the workload on the database. So, before making any database and resource environment changes, it necessarily must be optimized first.

SQL from the application will be limited by resource availability in the database itself. The logical input/output of the SQL workload may be artificially reduced by contention within the database from resource waits and queuing. Also, that may make the resource utilization look more favorable than it should be.

So we cannot use response time and physical input/output performance of the SQL for optimization purposes.



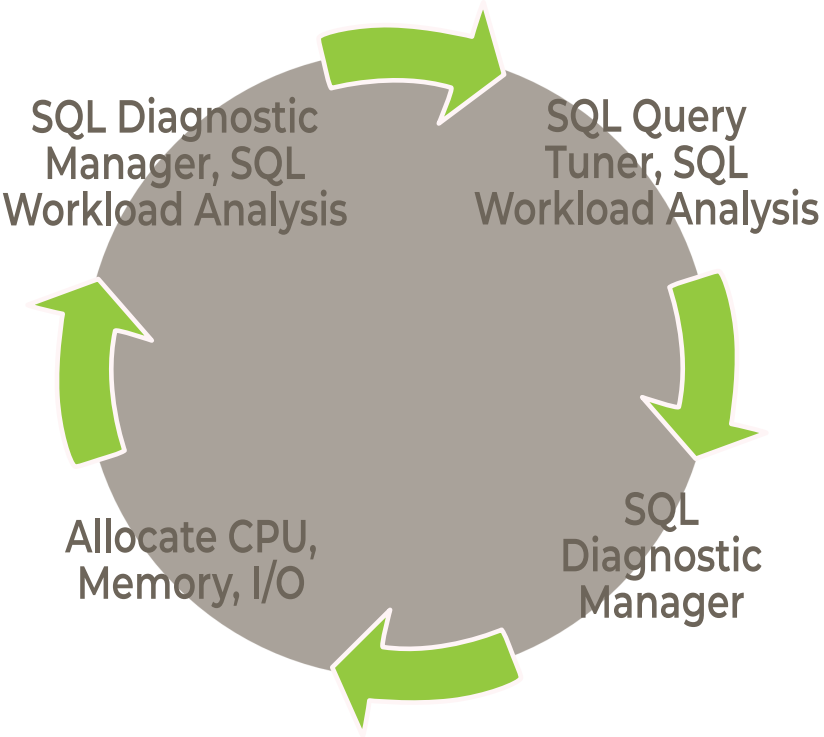
Contention causes waits and queuing. As such, inefficiency of application SQL can reduce the apparent SQL load.

Application logic and system design can undoubtedly help user performance. However, our primary goal here is to optimize the logical input/output measure, and logical reads/writes before attempting other changes.

The total workload is the concern here. Sothe worst-performing SQL is not necessarily the best place to start. Frequently executed SQL tuning can yield a more significant overall improvement than an infrequently called “worst SQL statement”.

TUNE THE DATABASE INSTANCE TO THE OPTIMUM SQL WORKLOAD

Now, we can tune the database instance in confidence to meet the optimized logical load of the application. Setting memory allocations, indexes, and storage distribution to increase performance now makes sense. It is important not to rely on database self-optimization features. While these automated capabilities can be beneficial, in some cases, auto-optimization can cause problems. Then, automatic actions may need to be overridden depending on the application and SQL.



This tuning process involves (1) monitoring for performance, (2) optimizing logical SQL, (3) optimizing the database, (4) allocating CPU, memory, and input/output, and then repeating these steps until achieving the desired results.

ALLOCATE RESOURCES

Only after the application and database layers of the application stack have been optimized should you determine the resource requirements for the desired performance for the end users. The optimization efforts will ensure that no money is wasted on hardware procurement and cloud usage fees. Also, the proper processor, memory, and storage requirements for the application

can be determined for the desired application performance and projected user load for the organization in question.

MONITOR AND REPEAT

The final phase of this user-centric database application tuning approach is the same as it was for most database administrator tasks: Monitor and repeat. Application loads change. And sometimes,, application loads increase dramatically with changes to the organization. Technology changes and new and more efficient resources, may come online that change the approach used for a particular application.

Continuous monitoring is needed, not just to catch anomaly situations, but to detect trends in changing performance. However, the systematic tuning of each layer must be maintained while repeating the process.

THE RIGHT TOOLS FOR THE RIGHT JOBS

Depending on the structure of the organization, a single team, or an individual may perform all of the tuning tasks through the application stack. However, in most cases, this work is distributed across multiple groups.

While a single, multiple-purpose tool might be used to help with all these tasks. It would certainly be overkill for specific development teams, for example.

Tools specific to tasks and teams means that developers and database administrators can focus on their tasks within this collaboration with user interfaces and capabilities appropriate for those tasks.

CONCLUSION

Do not wait until the environment is showing distress to implement the plan to optimize the performance of applications.

ABOUT THE AUTHOR

Scott manages IDERA's database performance management products. He has over twenty years of experience in product management and product marketing in the software and technology industry from small start-ups to Fortune 500 companies. For the past fifteen years, Scott focused on database performance and security products at various companies. Earlier in his career, he was a software engineer in the space and defense industry. Scott holds an MBA from Rice University as well as bachelor's and master's degrees in electrical engineering from the Georgia Institute of Technology.