Database Performance Monitoring

Buyer's Guide

October 16, 2016

VividCortex

Table of Contents

Executive Summary

More and more companies have begun to recognize database performance management as a vital need. Despite its widespread importance, good database performance management requires specialized expertise with custom approaches—yet all too often, organizations rely on one-size-fits-all solutions that theoretically "check the box" but in practice do little or nothing to help them find or prevent database-related outages and performance problems.

This buyer's guide is designed to help you understand what database management really requires, so your investments in a solution provide the greatest possible ultimate value. We've formatted this guide to aid you in weighing the available options and considering which product you should use to gain deep visibility into your databases. It aims to:

- Help you understand what characteristics and functionality you should look for in an effective monitoring solution.
- Provide a map of the specific functions available in various products in today's market.
- Assist you in comparing several monitoring products side-by-side, with an easily customizable and universally applicable worksheet.

The worksheet supplies a comprehensive if not exhaustive list of the specific features that differentiate one product from another—we hope this asset in particular will provide some clarity in the often difficult decision-making process.

With this guide on hand, you will have a clear and illuminating roadmap to help you explore the complex and growing database monitoring marketplace. Which solution should you choose in order to satisfy your organization's specific needs? What features will provide you the greatest value? This guide will equip you to answer with confidence.

I. A Database Monitoring Solution Should Be Database Focused

Databases are not like most systems. There's more to monitoring a database than graphing some counters and CPU utilization trends over time. We build modern apps by deliberately making most of the tiers stateless, which makes them easy to manage and scale. But that means we are delegating the heaviest lifting of all to the databases, where the statefulness is concentrated and specialized. Our apps then make highly demanding, almost arbitrary queries against the databases and assume they will perform well.

Any monitoring solution that is not database-centric, capturing the full scope and depth of database activity and instrumentation, is incomplete. To monitor databases usefully, tools must understand that databases are specialized, and that the most important dimensions of database performance don't fit into the same abstractions that serve other parts of the architecture.

Application performance monitoring (APM) products should not be treated as a substitute for specialized database monitoring. According to Gartner's definition of APM, "component deep-dive monitoring in application context" is the sole non-application-focused piece of required functionality. The phrase "in application context" should be considered carefully. While the bottom-line effect of a database on the application is the primary concern for most businesses, that does not mean you can focus exclusively on monitoring the application. In fact, the vast majority of real database incidents are collateral damage or caused by out-of-band traffic, which *has no application context* in the way some readers might assume because it doesn't come from the app. A classic symptom of such an assumption is when inexplicably slow application transactions may be identified, but not diagnosed or solved, from the APM tool. This is the all-too-common "visibility gap" from which many an APM customer

suffers, because APM vendors are focused on “end-to-end visibility.” Many of the most serious database problems arise in or near the database tier, and with the specialized nuance of that tier in mind, “end-to-end” does not truly exist.

II. Practical Considerations for Use, Configuration, and Optimal Value

Just as when purchasing any product, one of the first things to consider when shopping for a database monitoring solution is whether a particular option makes sense for you in practical terms. You’d think twice about buying a Hummer if your garage could only fit a single compact sedan; similarly, your database monitoring tool should comply with the strict logistical requirements implicit in your system. Unfortunately, unlike the more obvious constraints presented when car shopping, like garage size or the number of passengers you’ll need to carry, many of the requirements for monitoring products are elaborate, less visible, and dependent on other complex factors. Understanding what to look for from the outset will make your buying process more streamlined.

II.i. Scalability and Flexibility

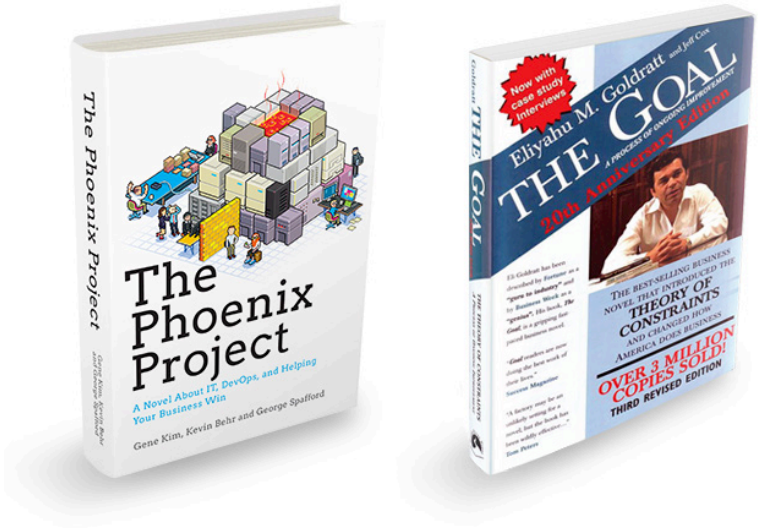
Modern applications can be dynamic and unpredictable, and monitoring solutions need to take that in stride. If your monitoring solution is to grow with you and support you without being a burden or a hassle, it ideally will have the following attributes:

- **Rapid deployment.** Instrumenting a new host should be a non-event; it should take a single shell command and a 10-second installation at worst. Ideally it should just require creating a role or tag in your configuration management system. There should be no burdensome dependencies (such as a JVM) or conflicts with system libraries.

- **High performance.** The monitoring tool should never impact the system being monitored. It should have no “edge cases” that will cause it to hog resources such as I/O, and it should self-monitor and self-limit to protect against unforeseen circumstances.
- **Self-maintaining.** Agent software should not require you to update or reconfigure it to get the benefit of new features or fixes.

II.ii. Built for Entire Teams

One of the biggest benefits of DevOps is the enormous productivity boost that comes from breaking down barriers and empowering everyone on the team to participate in solving hard problems. In fact, specialization and the team bottlenecks it can cause are primary themes of both *The Goal* and *The Phoenix Project*, books that are considered classics in the DevOps community.



When more people participate—say, a whole team—the resolution of database-related problems benefits greatly. The saying “Given enough eyeballs, all bugs are shallow” certainly applies. But despite the fact that most organizations have 10 to 50 times more developers than DBAs, most database monitoring products are built solely and inflexibly

for DBAs, not developers. To generate the biggest benefits from your organization's investment, a monitoring tool must:

- Give immediate and frictionless insight into production databases.
- Have a graphical user interface—not text-based—for productivity and clarity.
- Be designed and built with empathy for developers and awareness of their unique skills, other tools they like to use, and their workflow preferences.
- Have appropriate team productivity and collaboration features, such as deep linking and flexible role-based access control.

II.iii. Multi-Tenant SaaS Deployment

While some organizations still opt for and insist upon on-premise deployment, signs indicate that the future of monitoring is likely to be SaaS. Though on-premise solutions appear to offer a handful of advantages that are important to some buyers (such as customization or security), the advantages in these areas are actually superficial and short term. A measured, insightful view of the industry's evolution reveals that the benefits of multi-tenant SaaS solutions are driving faster adoption across the board, and most enterprises now spend significant portions of their IT budgets on SaaS solutions due to the competitive advantages and cost savings they can gain.

SaaS solutions are:

- **Always on, accessible anywhere.** Monitoring tools that live on someone's laptop aren't useful to other team members, and monitoring tools that go down with the ship don't help you solve problems when you need them most. Your monitoring should be independent of your internal systems.

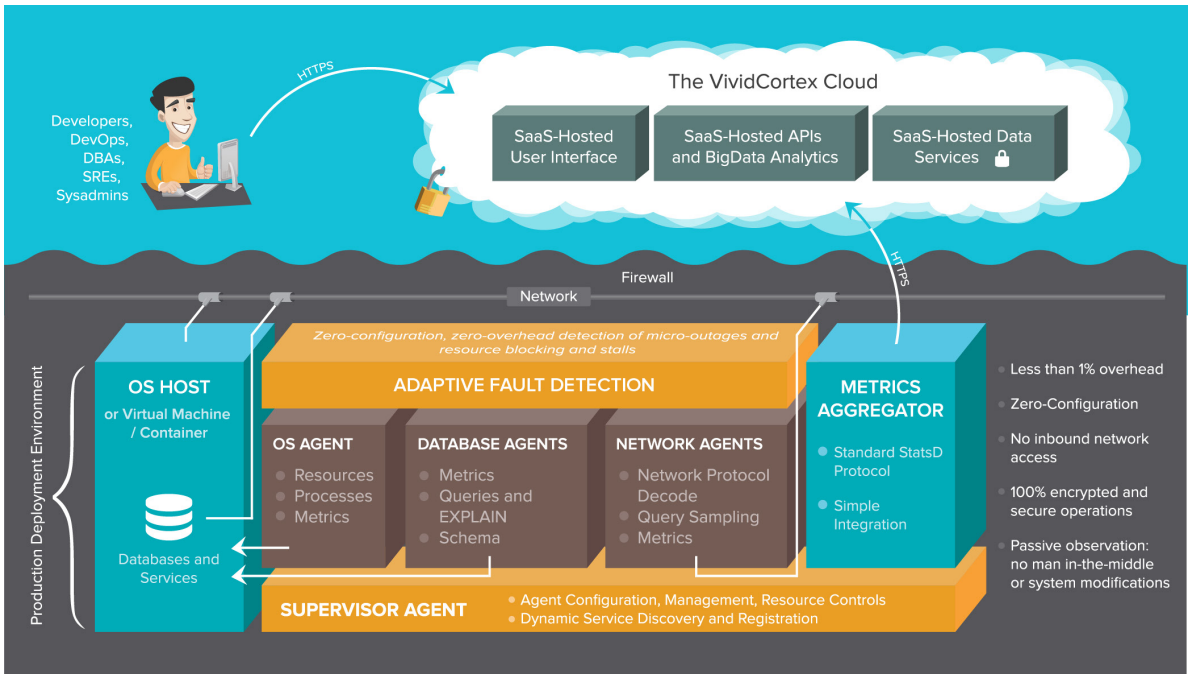
- **Dramatically less expensive and more nimble.** The costs of hosting your own monitoring tools include hardware; datacenter space; power and cooling; maintaining, upgrading, and securing the servers; making the servers highly available; and staffing. A multi-tenant SaaS solution lightens the burden in each of these areas by virtue of shared costs and economies of scale.

- **Continually innovating.** A SaaS-hosted solution receives continuous upgrades with new, better features, without requiring you to lift a finger. A multi-tenant solution enables shared learning and best practices, so you get "herd immunity" benefits from improvements created for other customers.

- **At least as secure.** Well-architected, purpose-built SaaS solutions can be at least as secure as—and often more secure than—on-premise alternatives. Do you have a team dedicated to securing and supervising your on-premise monitoring solution? If there's an internal breach, do you know what access your employees might have to sensitive customer data? Do open-source self-hosted monitoring solutions encrypt sensitive data in flight and at rest? In many cases, the answer to each of these questions is "no." The high degree of focus of a robust SaaS solution presents a much smaller, better-secured attack surface than most teams can achieve on their own.

- **Built by experts, by definition.** Any vendor who operates a large-scale SaaS platform has a better chance of understanding the challenges you face in building and operating your own applications.

As an example of how a database monitoring tool might fit into your systems, here is a blueprint of VividCortex's structure (a cloud-based, SaaS solution):



III. Specialization and Insights Specific to Databases

If your database monitoring solution is truly database focused, how does that advantage translate to specific capabilities? Keep in mind that these capabilities aren't mere bells and whistles—the features described here are more fundamental, and though some product-specific characteristics (as outlined in this guide's worksheet) may touch on these concepts, they also represent bigger philosophical differences in how a solution approaches its task and goals.

III.i. Query Monitoring and Workload Analytics

The single most specialized and important feature of any first-class database monitoring solution is that it puts queries (requests, statements) front and center. The database's sole reason for existing is to run a workload—to execute queries. Therefore, the only way to know if the

database is healthy and performing well is to measure what matters: whether its workload is completing successfully, quickly, and consistently. Everything else is essentially a vanity metric.

Query monitoring that's useful is deep and nuanced. Queries need to be measured overall as well as individually, and they need to be analyzed deeply. This helps answer questions such as:

- Are any categories of queries consistently performing badly or consuming too many resources?
- Are any categories of queries occasionally creating bad customer experiences, despite looking fine most of the time?
- Are there any queries that can be optimized to make the database run better overall?

A query monitoring solution has to be able to answer those and many other questions for users. This requires deeply intelligent product functionality that supports workload analytics as well as monitoring. Query workloads are huge and complex datasets. Humans cannot examine them manually, even at small scale; but when users identify issues, they need to be able to drill into the details and apply their unique intelligence and application knowledge to solve problems.

The best query monitoring and workload analytics solutions have features such as:

- Query aggregation and "tallest tent pole" slice-and-dice, to find big problems fast.
- Drill-down to individual statement executions, to examine the specifics.
- Automated capture and analysis of query execution plans (EXPLAIN plans).

- Automated textual analysis of SQL statements to find red flags and bugs.
- Workload change analytics (e.g., time-versus-time comparison, to reveal workload changes).
- Statistical tools such as regression, to find the relationships between metrics.

Many monitoring tools claim to support features such as “correlation,” but in reality do not. Establishing a correlation requires more than simply lining up graphs vertically and allowing a user to eyeball them. Sophisticated analytics solutions should be backed by substantial math and statistics.

III.ii. Deep, Effortless Drill-Down and Zoom-in

To monitor large, distributed, diverse systems effectively, the monitoring tool must present an aggregate view and enable rapid zoom-in and drill-down to the finest level of detail. Without the 50,000-foot view, monitoring isn’t scalable, but without the deep dive, you can’t solve problems effectively.

Ideally, a database monitoring solution offers:

- High-level summary views curated to surface the most important insights visually. Monitoring one server at a time does not scale and is all but useless.
- Rapid drill-down and zoom-in by arbitrary dimensions: hosts, tags, users, roles, time ranges, etc.
- Extremely fine detail at the lowest level. At least one-second granularity and microsecond precision are required, because problems usually start at sub-second durations and escalate, and millisecond-level timings are too coarse when query performance

is often sub-millisecond. You must be able to drill down to individual queries, disks, or processes.

- Sufficient granularity to show the resolution of *distributions*, not just averages, of important dimensions such as latency. These are usually exposed as histograms, quantiles (percentiles), and heatmaps. Metrics of percentiles are a red flag, because they're usually generated by an application of math that warps the true meaning of what is being calculated.

III.iii. Sophisticated Anomaly Detection

A proper database monitoring solution ingests far more data than you might expect. The volume of telemetry is typically several orders of magnitude greater than that generated by basic system monitoring, due to the high-cardinality, highly variable, dynamic workloads. Humans cannot begin to process this volume and variety of data. And there's no way to build dashboards for datasets with truly profound variety—intrinsically, a prebuilt dashboard can only show you what you already know, can only display problems you've seen before.

Traditional monitoring tools let you generate alerts on static thresholds, such as "CPU exceeded 90%." But this approach does not scale. Systems are constantly changing; what's normal at 2 am is very different from at 2 pm, and what was once a meaningful threshold can become unmeaningful with a shift in context. Static thresholds are proven to generate false positive alerts that desensitize users and rapidly result in teams learning to ignore their monitoring systems.

That's why all modern monitoring tools, of whatever stripe, offer some form of anomaly detection. This is the minimum requirement in order to be market competitive. However, for database monitoring it's particularly important, due to the variability and volume of the telemetry. Features you should look for in a database monitoring solution include:

- Anomaly detection on important types of queries, such as a query that suddenly becomes a “heavy hitter,” or an important change in a “heavy hitter” query.
- Automatic baselining and seasonality prediction at high granularity. Algorithms that can do this are well known in the industry.
- “Lateral logic” solutions for measuring what matters most, such as time-to-live (TTL) monitoring for predicting time-to-disk-full, instead of alerting on the disk becoming too full as it happens.

Although basic anomaly detection is widespread, most of the techniques are not suitable for truly large-scale high-resolution monitoring of high-cardinality metrics; they will still create far too many false positives, causing more problems than they find. They’re acceptable for monitoring basic metrics such as CPU usage, but not for query anomaly detection. You should ask vendors how their anomaly detection algorithms work at scale.

III.iv. Proactivity

Proactivity is a buzzword that’s seldom really delivered by any monitoring vendor. The definition of proactivity in database monitoring is the prevention of issues before they ever occur, and most monitoring vendors focus on quick after-the-fact detection, at best. There are, however, a few ways that a system can actually achieve proactivity:

- **Early-warning features.** Does the monitoring solution find budding problems that will escalate, and surface them when they’re too small for customers and engineers to notice, but still small enough to diagnose cleanly? Such problems have a “golden time” before they begin to get more serious and harder to diagnose due to the escalating effects. Early detection is key.
- **Detection in pre-production environments.** Is the monitoring

solution designed and priced for both pre-production and production deployment? Does it have workload analytics capabilities to surface important changes when deployments to pre-production environments are executed? Does it have textual analytics to highlight problems that won't show up in staging, but will cause issues in production? Finding problems in staging environments, before shipping to production, is a proven method for avoiding outages.

Reduction of firefighting is also key. The best way to make your team proactive is to connect the feedback loops, smash the silos, and get the whole team focused on preventing and solving issues. Without this shift in mindset, those who operate the systems in production are doomed to spend all of their time in reactive mode, battling problems they couldn't prevent. But when the whole team has visibility into production database behavior, DBAs are freed up for the more strategic work they should be doing, preventing tomorrow's problems instead of solving today's.

IV. Built by Experts

Database monitoring is so specialized, and databases are such delicate and vital systems, that they have to be built by experts. Someone who builds a monitoring tool for Database X and then bolts on "support for monitoring MySQL" will not understand the specifics of MySQL—what types of problems tend to occur, what red flags to look for, what not to do lest you bring down the whole server. This is not just an academic point; bad monitoring systems are frequent causes of outages.

When you talk to a vendor, do they use specialized terminology that stands out, representing familiarity with a different crowd? Do they have a long history of contributing to and caring about your community? When you search for information on performance-related topics for a specific database type, do high-ranking results appear authored by members of that organization?

Another way to tell if the vendor is really competent is to ask what level of support you'll get from them. The answer indicates much more than just the level of customer service. Do they employ well-known experts with years of experience scaling databases and applications, or is their expertise only in writing monitoring systems? Are they running a large-scale system themselves, and do they have empathy for the challenges you face? Are you thrust into an online community full of the blind leading the blind? Will the vendor really understand what their own product is showing you about your specific databases? Can they teach you things you didn't know about your application? Or is their attitude "We just build it; knowing how to use it is your job"?

V. Choosing the Right Approach: A Few Examples

Below are several examples of common database-related frustrations, along with potential monitoring-related fixes. While we recognize that "frustrations" in complex systems very rarely (if ever) have a single root cause, these examples are emblematic of issues that database performance management is designed to address. These samples can give you a brisk idea of how the right solution can positively impact your situation.

Common frustrations:

1. **Poor app performance and availability.**
    a. *Caused by:* Server stalls, faults, bad query performance, poor latency, etc.
    b. *Addressed by:* Deep visibility into performance metrics, allowing users to find the issues and nonoptimal operations in the system.

2. **Customers reporting performance issues before they're discovered internally.**

    a. *Caused by:* Insufficient ability to see performance metrics with depth and at scale.

    b. *Addressed by*: A complete view of database activity, tailored to monitor the most important and indicative elements of a given system.

3. **Engineers' dependence on DBAs for releasing code.**

    a. *Caused by:* Inability to self-service and independently see code's effect on the system because of unfamiliarity with the database.

    b. *Addressed by:* A platform that provides ready access to and decipherability of the systems to many types of users and teams.

VI. Choosing the Right Solution: A Customizable Worksheet

A number of essential components should be considered before investing in a performance management solution. This comparison chart will help you evaluate solutions you may be considering.

Solution 1:

Solution 2:

SETUP AND MANAGEMENT		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Access controls with roles and permissions	☐	☐
Secure user login	☐	☐
Low cost	☐	☐
Platform is managed by you	☐	☐
Data collection within minutes	☐	☐
Dynamic scaling—add users and instances as needed	☐	☐

END-USER EXPERIENCE		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Cloud-based for easy and secure anytime/ anywhere access	☐	☐

END-USER EXPERIENCE		
Intuitively designed interface and functionality	☐	☐
All major browsers supported	☐	☐
Designed for instinctive interaction to enable self-service, fast adoption, and ease of use	☐	☐
Able to access and manage different environments	☐	☐

ARCHITECTURE		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Secure, cloud-hosted platform	☐	☐
Multi-tenant deployment	☐	☐
REST API, not SOAP	☐	☐
Proven 99.8% uptime reliability	☐	☐

COLLABORATION & SHARING		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Unified view of database systems	☐	☐
Deep linking (shareable URLs)	☐	☐
Configurable alerts	☐	☐
Alert integrations: HipChat, PagerDuty, Slack, etc.	☐	☐
Unlimited users within or across teams	☐	☐

REPLICATION		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Monitor replication lag	☐	☐
Start/stop events for replication	☐	☐
Configurable alerts on replication delay/lag	☐	☐

ANALYTICS		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
One-second, deep drill-down	☐	☐
Automated metric and query anomaly detection	☐	☐
Automated stall detection	☐	☐
Identification of new and important queries	☐	☐
Query samples and EXPLAIN plans	☐	☐
Index recommendations	☐	☐

HISTORICAL DATA TREATMENT		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Historical and trend analysis	☐	☐
Comparison and customization of time frames	☐	☐
Configurable data retention	☐	☐

SECURITY		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Detection of SQL injection attempts	☐	☐
In-flight and at-rest encryption	☐	☐
Role-based access control	☐	☐
Single sign-on (SSO) via SAML	☐	☐
Auditing compliance	☐	☐

OPTIMIZATION		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Query perspective	☐	☐
Work metric perspective	☐	☐
Performance metrics	☐	☐
Custom queries	☐	☐
Deadlock metrics	☐	☐
Easily identifies top sources to load	☐	☐

WORKFLOW		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Integrations	☐	☐
Customizable alerts	☐	☐

WORKFLOW		
Role-based access control	☐	☐
Weekly and daily summary reports	☐	☐

CUSTOMER SERVICE & SUPPORT		
HIGH-LEVEL FUNCTIONAL REQUIREMENT	Solution 1	Solution 2
Health reports	☐	☐
Responsive support team	☐	☐
Easy access to engineers	☐	☐
Scheduled training sessions	☐	☐
Customer updates	☐	☐

Total selected:	
0	0

About VividCortex

VividCortex is the best way to improve your database performance, efficiency, and uptime. It is a secure, cloud-hosted platform that eliminates your most critical visibility gap, providing deep insights into production database workload and query performance. Unlike traditional monitoring solutions that focus on vanity metrics or aspects of performance management that capture only a superficial, app-centric view, VividCortex measures the performance and resource consumption of every statement and transaction, then uses patented algorithms to analyze and surface relevant insights so you can proactively fix future performance problems before they impact customers.