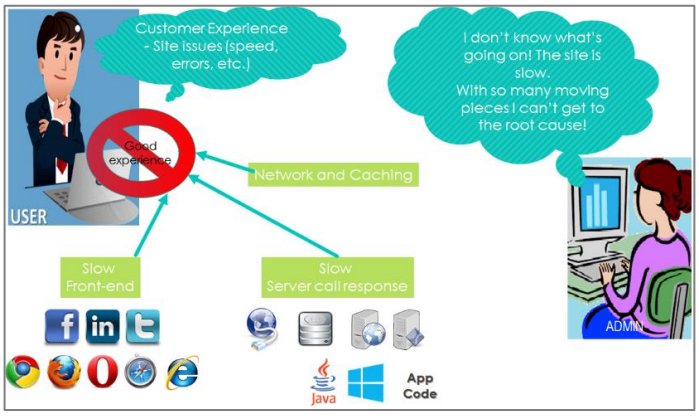
Diagnostics in Testing and Performance Engineering

This document talks about importance of diagnostics in application testing and performance engineering space. Here are some of the diagnostics best practices to ensure enterprises are able to detect and isolate issues early in the life cycle or identify the root cause and fix the issues quickly in the production environment, minimizing costs and averting risks.

Introduction

There've been plenty of examples around how an e-commerce site launching a new phone had difficulties in maintaining the site available for hundreds of thousands or even millions of people around the globe.

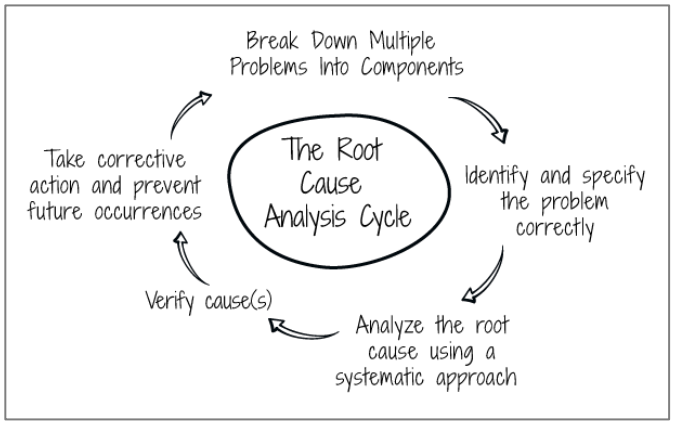
The challenges in terms of volume and speed are growing exponentially and so, is the load on an already overloaded enterprise. It's not just with e-commerce, it could be a normal banking site or a mobile banking application which fails to execute transaction seamlessly due to slow server response leaving an unsatisfied customer.



What causes an unsatisfied customer and loss of revenue, is the inability to diagnose issues properly so that they can be fixed early in the cycle.

What is diagnostics?

Diagnostics is a science that determines the root cause of the issue. For example, an issue could be that the application is not letting user to add an item to the shopping cart.



This could be because of an unexpected behavior of the application or the backend systems, or delays or packet losses owing to the network, or may be slow browser rendering because of an unresponsive script. The root cause can be anywhere.
The difference between an average enterprise and a 21st century customer-centric enterprise is the ability to pin-point the root cause of an issue to fix it early in the cycle.

Risks and costs associated with not doing proper diagnosis are huge. In today's competitive world, this is what differentiates success and failure.

From performance testing and engineering perspective, diagnostics finds itself in between Quality Assurance and Operations Performance Validation.

Minimum Required Capabilities

They say, "A True Diagnosis Is Half the Cure", hence to achieve true diagnosis, here are some recommendations.

Since issue can be anywhere, enterprise must have an infrastructure that provides ability to diagnose:

- The Server side – the back-end,
- The Network side, and
- The User side or the front-end

This means that a monitoring agent should be available at these places within an enterprise to capture data and send it to the analytical engine to infer and obtain decision making insights. This should be fast and seamless, hence these components should be tightly integrated.

Diagnostics Best Practices

Once the infrastructure is in place, the big question is what to look for when diagnosing the root cause. It's similar to a doctor taking vital signs, doing an assessment of symptoms and correlating it with external objects, parts or specific organs of the body to determine what is causing an unhealthy state. They have their own diagnostic codes and corresponding procedures; here we've put together some best practices around diagnostics to help you identify key areas to look for when diagnosing an issue.

Over the years, we've worked closely with our customers in numerous industries and domains and have innovated our approach towards performance and diagnostics along with their quest for exceptional customer experience and digital transformation. Following parameters individually or in combination could cause issues and this needs to be monitored and examined on priority. One issue may lead to another issue elsewhere within the enterprise. Issues may relapse when root cause is not properly done.

Server Diagnostics

Let's explore diagnosing the symptoms for an unhealthy application at the server side.

CPU usage: Diagnose whether the CPU usage is because of an application instance, some other program, or batch jobs. Determine whether the CPU time is in user or kernel space. This will help us focus on two things, whether Java instance is consuming CPU or CPU is high because of some other parameters like garbage collection, batch jobs, Disk I/O, or high memory consumption etc.

Load average: Find out whether the high load is due to CPU consumption by the application or due to disk I/O. This will help in determining whether the application design should be re-evaluated or that the disk needs to be partitioned appropriately for usage by other systems and users.

Virtual Machine Monitor: Quickly check the health of your VMware host by monitoring CPU, memory utilization, number of virtual machines configured and running, and much more. Any degradation in health of one or more VMs can impact customer experience.

Load balancing: Evaluate load balancing across multiple data centers and between servers. Erratic load balancing can cause a server to go down because of unprecedented high load.

Thread Dump: Using a snapshot of java thread dumps, analyze and verify the state of threads, whether they are running or stuck (wait or deadlock). For example, if a thread is stuck on a particular method, then it affects the server response time. Thread dumps can be:

- Alert based,
- On-demand, and
- Schedule based

Heap Dump: Depicts the usage of memory by different java objects and determine memory leak or any other heap issues. This can be due to code and design of the application, which leads to memory leak and GC is not able to clear up the unused objects from the heap, leading to frequent GCs and subsequently high CPU to make server unresponsive.

Flight Recorder: Capture detailed low level run-time information on how JVM and java applications are behaving. This is a must have for after-the-fact incident analysis to determine the behavior of the application at the incident when the issue occurred.

Java Garbage Collection Tuning: Analyze groups and members for garbage collection memory used. By tuning java parameters, the performance of the application can be enhanced. For example, adopting from several GC policies whichever suits the application for optimum performance. Analysis can be tuned for:

- Heap Size
- Young, tenure and permanent generation size
- Garbage Collection policies can be defined

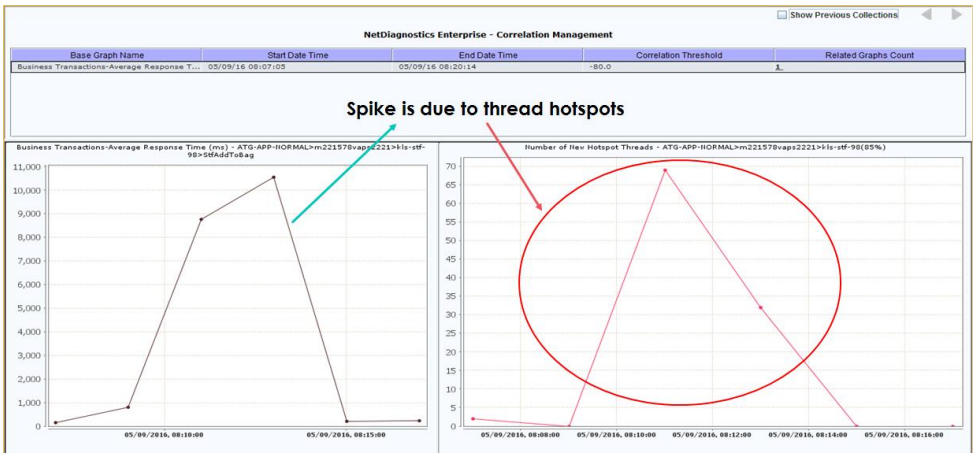
Methods: Analyze methods to determine issues at the code level. Methods can be analyzed for

- Method level invocation rate and execution time
- Comparison with baseline before and after the fix

DB Queries: Database is another critical service side resource that could be the root cause of the issue. Analyze DB queries for number of active connections, CPU, queries, and lock. This will lead to a conclusion whether DB is a cause of slow response. This, upon fix, can be compared for optimization.

Business Transaction Health: Analyze business transactions whether they are slow, very slow, or are causing errors. This can be viewed right at the tier view level and can be segregated for execution and response time. This helps us to narrow down our focus to specific business transactions and related server side components.

Hotspots: Identify the outlier metrics and pin-point the root cause of the slow response. Hotspot highlights where ever there is too much waiting in the response or a blocked-response. This information points to specific thread or method which is the cause for the slow response time.



Pattern Matching: Analyze two or more members or measurements in the groups showing the tendency to vary together, contributing to the issue.

The root cause can be determined by correlating spike or matching patterns with other metrics. For example, the high response time could be because of a thread hotspot.

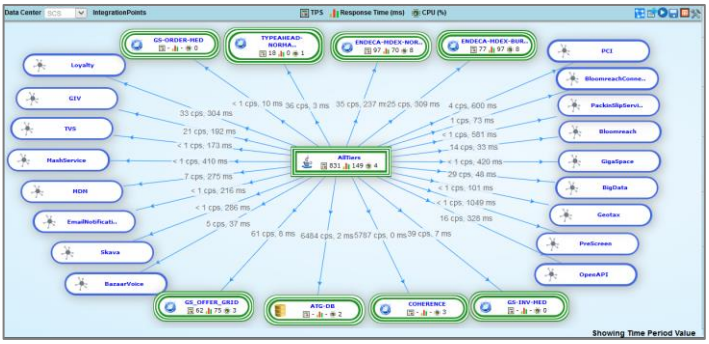
Thread pool: Analyze the threads as per configuration, what are they doing, along with their status – hogging or in blocked state. Thread pool gives information about idle thread count, total thread count that the application can handle, and pending user requests. This provides insight into the root cause of high response because of thread's performance.

JDBC Pool: Analyze the database related information like DB connection, leak connection count, etc. This pin points to the issues related to the database server.

Coherence: Analyze cache hits/miss, health stats, cluster stats, service stats to determine the bottlenecks. For example, if the product is not cached in coherence, then the requests directly go to the DB, which is considered as overhead to the performance of any application and may lead to slow server response. Coherence cache needs to be updated periodically via batch jobs.

Integration point: Monitor and analyze availability and performance of the integration points, its thread pool usage, and tuning. Get early warning before the threads used by an integration point get busy. For example, a payment call may take time to execute because of issues related to payment integration points (PCI server)

Log Analysis and Monitoring: Application and server logs can be analyzed to determine root cause of the issue. The entire scenario can be recreated from a forensics stand point.



Integration point monitoring

Network Diagnostics

Diagnosing the symptoms for issues that may be arising because of problems at the server side.

Network layer errors/re-transmission: Analyze server for the limit of max connections on a simple IP. This may lead to finding issues related to the need of adding more IPs if required.

Cache diagnostics: Analyze whether the network cache is offloading properly or not. This allows to determine whether the network cache is able to manage appropriate traffic and divert the rest of the traffic to the origin server.

Client-side Diagnostics

Browser level diagnostics: Analyze browser behavior. Following can help determine root cause of the issue. Alternatively, these can be mapped to the server diagnostics and findings to pin point a root cause.

- Analyze Number of resources used in a page. Is caching used efficiently?
- Analyze page score to determine potential performance improvement areas like minimizing of CSS, reduce redirects, and so on.
- Analyze impact of third party tags like FB, Twitter, etc.
- Analyze the impact of analytics tools tags like Omniture.
- Analyze the number of trackers used.

Alerts: Alerts are extremely useful in providing early warning before the performance degrades. Alerts help in taking or automating a corrective action including collection of relevant data which would help in diagnostics, such as thread dumps. Alerts can be of following types:

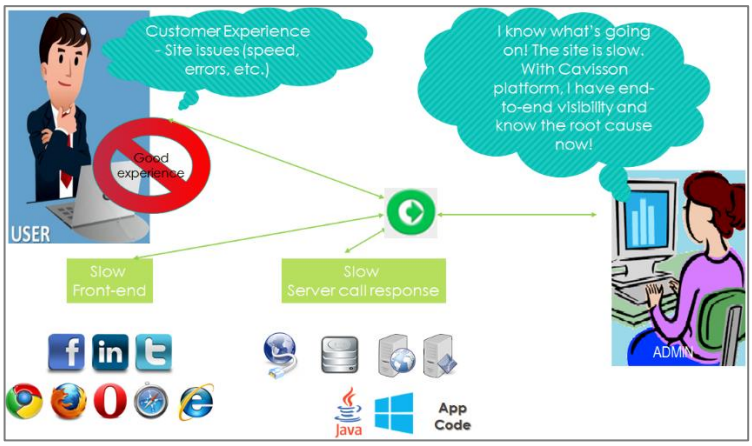
- Capacity alerts based on SLAs.
- Behavior alerts based on baseline or the trends of the previous performance.

How are these capabilities delivered?

All the above capabilities can be delivered by Cavisson NetDiagnostics Enterprise (NDE). NetDiagnostics Enterprise is a complete Application Performance Management (APM) solution for both lab and production. It offers simple and intuitive view of live application traffic and all contextual analysis with respect to the above features.

NetDiagnostics offers deep diagnostics at various levels, such as class, parameter, and method. Users obtain complete visibility of distributed applications with auto-discovered end-to-end business transactions, baseline performance alerts to quickly isolate and resolve issues.

World leading retailers, financial services organizations, and network operators, rely on Cavisson for quality, performance, and availability of their mission critical applications.



To schedule a review of your performance engineering initiatives, contact us now at sales@cavisson.com