Monitor Java application through an agent

Vipul Saini
vipul.saini32@gmail.com
Meerut Institute of Engineering and Technology, Meerut, Uttar Pradesh

Dr. Ajay Singh
ajay41274@gmail.com
Meerut Institute of Engineering and Technology, Meerut, Uttar Pradesh

ABSTRACT

A typical application environment consists of the different components that interact in a variety of ways to fulfill requests from the application's users: Web applications served from an application server, Databases or other data stores, Remote services such as message queues and caches. Our app agents automatically discover the most common application frameworks and services. Using built-in application detection and configuration settings, agents collect application data and metrics to build flow maps. A flow map visually represents the components of our application to help us understand how data flows among the application components. For example, the business transaction flow map for a simple e-commerce application below shows data flowing between web services, message queues, and databases.

Keywords: Business Transaction, Application Monitoring, Agent, Method Level Monitoring, Thread Level Monitoring, Database Visibility, Infrastructure Visibility, Tiers, Servers, Backends.

1. INTRODUCTION

1.1 Business Transactions

A business transaction represents the data processing flow for a request, most often a user request. In real-world terms, many different components in our application may interact to provide services to fulfill the following types of requests:

In an e-commerce application, a user logging in, searching for items, or adding items to the cart.

In a content portal, a user requests content such as sports, business, or entertainment news.

In a stock trading application, operations such as receiving a stock quote, buying, or selling stocks.

Agents discover requests to our application as entry points to a business transaction. Similar requests, such as user log in, are treated as multiple instances of the same business transaction. The agents tag the request data and trace the request path as it passes from web servers to databases and other infrastructure components. Agent collects performance metrics for each tier that processes the business transaction.

Because Agent orients performance monitoring around business transactions, we can focus on the performance of our application components from the user perspective. we can quickly identify whether a component is readily available or if it is having performance issues. For instance, we can check whether users able to log in, check out, or view their data. we can see response times for users, and the causes of problems when they occur.

1.2 Java based Business Applications

A business application is the top-level container in this model. A business application contains a set of related services and business transactions.

In a small scale deployment, only a single business application may be needed to model the environment. In larger deployments, we may choose to divide the model of environment into several business applications.
The best way to organize business applications for us depends on our environment. A leading consideration for most cases, however, is to organize business applications in a way that reflects work teams in our organization, since role-based access controls in the Controller UI are oriented by business application.

1.3 Servers

A Server in the model corresponds to monitored server or JVM in the application environment. A server is the smallest unit of the modeled environment. Depending on the agent type, a server may correspond to an individual application server, JVM, CLR, PHP application, Apache Web server.

Each server identifies itself in the model. When we configure the agent, we specify the name of the server, tier, and business application under which the agent reports data to the Controller.

1.4 Tiers

A tier is a unit in the model is a grouping of one or more server. How we organize tiers depends on the conceptual model of our environment.

Often a tier is used to group of a set of identical, redundant servers. But that's not strictly required. we can group any set of servers, identical or not, for which we want performance metrics to be treated as a unit into a single tier.

The single restriction is that all servers in a single tier must be the same type. That is, a tier can't have mixed types of agents, such as both .NET and Java servers.

The traffic in a business application flows between tiers, as indicated by lines on the flow map, which are annotated with performance metrics.

In the model:

There is no interaction among servers within a single tier.

An application agent server cannot belong to more than one tier.

1.5 Backends

A backend is a component that is not instrumented by an our agent but that participates in the processing of a business transaction instance. A backend may be a web server, database, message queue, or other type of service.

The agent recognizes calls to these services from instrumented code (called exit calls). If the the service is not instrumented and cannot continue the transaction context of the call, the agent determines that the service is a backend component. It picks up the transaction context at the response at the backend and continues to follow the context of the transaction from there.

Performance information is available for the backend call. For detailed transaction analysis in for the leg of a transaction processed by the backend, we need to instrument the database, web service, or other application.

2. APPLICATION MONITORING TYPE

2.1 Infrastructure Visibility

Infrastructure Visibility provides end-to-end visibility into the hardware and networks on which our applications run. we can use Infrastructure Visibility to identify and troubleshoot problems that affect application performance such as server failures, JVM crashes, and hardware resource utilization.

2.2 Browser Real User Monitoring

When we add End-User Monitoring to Application Performance Management, we can correlate business transaction performance to the user experience for those transactions.

If server app agents run on the applications that serve our browser applications, we can further configure the app server agents to inject JavaScript agent into the code that runs on the browser. Access the settings to configure injection in the Applications Configuration page.

2.3 Database Visibility

In Application Monitoring, a database called by an instrumented server is considered a remote service. we can get a significant amount of information on the interaction between the application server and database, but not from the database server's perspective. When using Database Visibility with Application Monitoring, we can drill down to detailed database performance information directly from application flow maps.

2.4 Method Level Monitoring

In method level monitoring, Agent capture the time taken by executed method for serving a request.
2.5 Thread level Monitoring

In thread level monitoring, the agent detects which thread is stuck or taking more time.

2.6 Log Analytics

For those times when tracing application code doesn't provide enough clues to track down the cause of a problem, our platform provides visibility into the transaction logs that can be correlated to specific business transaction requests.

3. PROPOSED WORK

We deal with following application monitoring type.

3.1 Infrastructure Visibility

There are three classes of Infrastructure Visibility functionality:

- Basic hardware metrics from the server’s OS, for example, %CPU and memory utilization, disk and network I/O
- Custom metrics passed to the Controller by extensions
- Run remediation scripts to automate our runbook procedures. We can optionally configure the remediation action to require human approval before the script is started.
- Run JVM Crash Guard to monitor JVM crashes and optionally run remediation scripts
- If we have a Server Visibility license, the Standalone Machine Agent provides the following additional functionality:
- Extended hardware metrics such as machine availability, disk/CPU/virtual-memory utilization, and process page faults
- Monitor application servers that run inside Docker containers and identify container issues that impact application performance
- The Tier Metric Correlator, which enables us to identify load and performance anomalies across all servers in a tier
- Monitor internal or external HTTP and HTTPS services
- Group servers together so that health rules can be applied to specific server groups
- Define alerts that trigger when certain conditions are met or exceeded based on monitored server hardware metrics
- Network Visibility monitors traffic flows, network packets, TCP connections, and TCP ports. Network Agents leverage the APM intelligence of App Server Agents to identify the TCP connections used by each application. Network Visibility includes the following functionality:
- Detailed metrics about dropped/retransmitted packets, TCP window sizes (Limited / Zero), connection setup/teardown issues, high round trip times, and other performance-impacting issues.

3.2 Browser Real User Monitoring

In browser real user monitoring, the agent detects the request on the bases of called method, this method can be called a entry point.

3.3 Method Level Monitoring

In method level monitoring, we make a .xml file in which we mention the package or class or method which we want to instrument. Agent do byte code instrumentation of these method.

Add following thing in method at runtime-

MethodStartTime
MethodEndTime
TotalTime = StartTime - EndTime

By this we can get total method execution time.

For Instrumentation Agent use transformer class.
4. EXPERIMENT RESULTS

We did a test, after applying the Agent on the java based application, we collected some metrics, which is shown below.

Chart -1 database visibility

Chart -2 Infrastructure Visibility
Chart -3 Browser Real User Monitoring

Chart -4 Method Level monitoring
4.1 Experiment Configuration

We can monitor stand alone java program or a browser based java application-
For Monitor java program/browser base
Add following Argument in JVM Argument
-javaagent:/root/weblogic/netdiagnostics/lib/ndmain.jar=time,ndAgentJar=/root/weblogic/netdiagnostics/lib/ndagent-with-dep.jar

4.2 Result Analysis

We can see from above result-

JDBC database is called
Url- /nsecom/HTTP_CALLOUT
Response Time taken by whole request- 653 mili second

On the bases of business transaction response time, we can decided business transaction is slow or very slow. We decide a threshold time, if business transaction cross that time, it will be come in slow/very slow. Business transaction can be debug further with help method calling tree to find, which method is taking more time.

Threshold time- lower limit- 500 ms, upper limit- 1000ms

So this business transaction will fall in slow category.

Now business transaction can be drill down to find which method is taking more time, with the help of method calling tree , it can be identified.

5. CONCLUSION

The Agent platform enables you to monitor and manage your entire application-delivery ecosystem, from the mobile app or browser client request through your network, backend databases and application servers and more.
Agent platform gives you a single view across your application landscape, letting you quickly navigate from the global view of your distributed application right down to the call graphs or exception reports generated on individual hosts.

6. REFERENCES