Application Audit

Michael Sydor

SERVICE ASSURANCE - APM

Scope

An *Application Audit* is a structured review of performance characteristics of an application while under synthetic load or live in production. It is a summary confirming correctness of the monitoring configuration, visibility into the application behavior and identifying the key components that best represent the overall availability, performance and capacity of the monitored application. When performance problems are identified, the Application Audit may also include specific recommendations, where the experience of the APM practitioner can contribute. The minimum goal is to simply summarize the overall performance characteristics and indicate what should be monitored in order to have a good production experience.

Duration

An Application Audit may be completed in a few hours or may take a week or more, depending on the number of components to examine and the availability of load generation or suitable production data. In general, a *pre-production Application Audit* should take 2-5 days, depending on the availability of load generation. A *production Application Audit* will take 2-4 weeks, depending on the consistency of the production experience, day-to-day and week-to-week. This best practice will show you what to do with your on-site time and how to summarize the findings in a consistent fashion.

Benefits

Completing an *Application Audit* is important to the short-term success of an APM initiative and is a valuable exercise to help you improve your relationship with the client. Showing how to use APM technology to support a meaningful exercise, like this audit, helps the client to appreciate the benefits of using APM techniques across the application lifecycle.

- Confirm a compatible and efficient monitoring configuration
- Confirm visibility sufficient to triage performance problems
- Characterize availability, performance and capacity metrics that are unique for the application
- Identify appropriate thresholds, alerts and present these as part of dashboards and reports
- Demonstrate value of the APM solution
- Provide an example of the basic use of APM technology in an easily reusable process
- Provide a foundation for efficient comparison of application releases and configuration changes
- Plan for short-term and long-term APM success

PRE-WORK

Your Preparation

The Application Audit is the bread and butter of APM Best Practices. If you cannot complete an audit then you really need to take more time with the product documentation or complete more pilot exercises! An audit is really just an extension of a pilot, but having more emphasis on realistic load generation, agent configuration tuning and generating a report of findings. And unlike a pilot, the audit is spending much less time demonstrating features and capabilities of the APM technology. With an audit we are exercising the capabilities of APM technology to do sometime meaningful towards understanding the performance characteristics of the application. We are confirming that the monitoring configuration is going to be suitable for production usage and will enable efficient triage, should performance problems be encountered.

Always try to make the Application Audit a mentoring opportunity. This activity is simply the best vehicle to get clients trained and effective in using APM technology to manage application (and software) quality. Because you are not under stress as during an outage – you can take your time and really learn how to use APM visibility.

Interpreting performance data takes practice. AVOID DRAWING CONCLUSIONS. Your job is to get *visibility* into the performance characteristics. This is not a bug hunt. You will not have any *detect-fix-retest* cycles with which to confirm any conclusions – the client is simply not ready for that level of interaction. Before they can commit to rapid repair and retest you have to first convince them that APM can expose the performance problems.

When a client is ready to actually fix and retest, by following recommendations that someone makes, the Application Audit processes actually becomes a *Performance Tuning* – basically a series of application audits with developers immediately confirming and fixing performance issues, followed by additional testing to confirm the benefit – or backing out unsuccessful repairs. This is a serious commitment of resources and far beyond the scope of a basic Application Audit.

You will also be able to use the Application Audit process to spot check application or software quality, in addition to confirming that the monitoring configuration is correct and complete. Ultimately, this is what clients need to do in order to achieve true, proactive management of application performance. But it can be a useful stepping stone to help a client understand the utility of APM visibility, validate a release or even plan to migrate a service to the cloud.

Setting Expectations and Scope

Sometimes an Application Audit is performed as a stand-alone service engagement. This is the original motivation for the process. Sometimes the Application Audit will be used to illustrate some of the APM Best Practices prior to a client committing to a service bureau program. Both activities are outside of the traditional sales cycle and should only be directed towards clients who have a functioning APM environment. Committing to audit an application, in a pre-sales situation, is a risky proposition because you likely will not be able to manage the client expectation correctly. They will instead be more interested in you "finding a problem" rather than showing the benefits of additional visibility – and you will delay the potential sale.

The Application Audit is more often a post-sales activity that can be completed in about 2-4 days. The client will have been operating an APM solution for at least 6 months. Sometimes it will be a few years before an opportunity to deliver an Application Audit presents itself. The sooner, the better, as it is critical to demonstrate "value" for the APM solution, before it gets incorrectly escalated as a product problem. The audit is a key process to exploiting APM and it is often completely absent in the client's experience. The audit is the best way to expose the gap between client ambition and current reality – which may be considered a gap in their processes with APM. Saying "Let's make sure you know how to audit an application and get ready for production monitoring." Sounds a lot nicer than "Are you really sure you know what you're doing?"

QA Audit - Week

If you are tasked with doing an Application Audit over five days, it will usually be because the environment is small (8 JVMs) to medium scale (24 JVMs). A couple of clustered applications and you will have a lot of data to review and perhaps a lot of adjustments to the testing schedule. QA changes and testing are usually done during normal business hours, so you can expect to get good amount of work done. Sometimes the testing can only proceed after hours, usually because other applications have priority. Make sure to find out which situation you are entering and make appropriate adjustments to the schedule.

Production Audit - Couple Weeks

An Application Audit over a couple of weeks is most likely going to be required if a QA environment is not available. This is much harder to complete because of the variation in production data as well as the presence of incidents that can distort the application performance. All these factors need to be accounted for. You need at minimum three 4-hour periods of consistent application behavior. The same time of

day, and hopefully the utilization is consistent, from one day to the next (actually, pretty rare). This is why production audits take so much longer – it can be weeks before the data is reasonably consistent.

However, when a production environment has been continuously monitored, *with a suitable monitoring configuration*, it is highly likely that you will have all of the historical information to support an audit in the SmartStor repository. So now the audit can be delivered in a couple days to one week, for a single application. You will not have this circumstance very often because most folks do not know how to confirm or *validate* that the agent configuration is suitable (has good visibility). So you cannot assume that you have correct visibility or even appropriate data retention in the SmartStor. So either setup a remote session to review the APM configuration and confirm that the data is there and useful – or plan for a couple of weeks!

Cloud Migration and SLA – Couple Weeks

In order to establish an SLA (where the A=Agreement) for cloud services, or to establish a migration priority, you will need to audit up to a dozen candidate applications. All of these would have lifecycle monitoring in place so that there would be plenty of historical data on which to draw. And while the bulk of this analysis will follow the Application Audit process, you will also need a fair amount of Assessment process (Application Survey and Incident Analysis) in order to fully justify the cloud migration strategy. Please see Chapter 3 – Assessments in APM Best Practices for details.

Developing an SLA is something you can expect from a client with good (and well documented) APM capabilities. You know that they have appropriate visibility and retention practices, and reasonable thresholds and alerting in place. They may not have the correct perspective in determining which metrics are critical for managing the relationship with the cloud provider. They may need assistance aligning performance measures that the cloud provider prefers. You goal, in this SLA-focused audit, is to adjust the monitoring configuration, validate the new metrics and reporting and guide both the client and cloud provider into a mutually beneficial agreement.

Best Practice Pilot – Couple Weeks

For existing clients, who have been operational with APM for at least six months, you can also use the Application Audit as a pilot to demonstrate how a Best Practice initiative might impact the client organization and help them to get more value and collaboration from the APM investment. When you find (via Skills Assessment or APM HealthCheck) that clients are only using default agent configurations and little or no use of dashboards and reports – then this is an excellent opportunity to show them what the Application Audit can do to address those gaps and help them to get more from their APM investment.

Clients are always struggling with staff training and retention and will often concentrate APM responsibilities onto a single, highly proficient individual. This is a reasonable starting point but if efforts are not undertaken to distribute the responsibilities among the various stakeholders, across the application lifecycle, the whole initiative can suffer and sometimes stall. Following the APM Best Practices can help the client organization recover but sometimes it takes a pilot activity to really drive home both the value delivered as well as confirming that the organization can "step up" and share responsibility.

This will be a *mentoring engagement* with the Application Audit tasks divided among the stakeholders. This allows each stakeholder to understand their responsibilities and contributions to the Application Audit, now delivered as a community effort, rather than the effort of a single practitioner. Here is a summary of a typical 'division of audit responsibility':

Stakeholder	Application Audit				
	Configuration Baseline	Application Baseline	Performance Baseline	Configure Thresholds and Dashboards	Triage
Dev	JVM tuning	Custom tracing			Level 3
QA		Use case visibility	Key Components - Reporting		
Operations			Alert Integration	Pre-production review	Level 1
App Owner			Acceptance Criteria	Pre-production Review	Level 1, 2, 3
Web Admin	Agent deployment		Key Components – Metric Groups	Thresholds, Dashboards, Reports	Level 2, 3

TABLE 1: COLLABORATIVE SUPPORT OF APPLICATION AUDIT

Basically you conduct the audit, in a mentoring role, mediating with the various stakeholders as you undertake and complete the audit tasks. Use the concept of an "Audit Request" and highlight the interfaces, where requests or data are exchanged. Capturing dialog in a word doc is an easy mechanism to capture these interactions and document the tasks and findings of each participant. The effectiveness of the pilot is in exposing the various internal processes and detailing what each stakeholder is responsible for. An Architect or Application Owner will be the 'document holder'.

Load Generation

The success in conducting an audit pre-production (usually QA or S&P: stress and performance) is a direct relationship to the ability to generate a consistent load against the application. How to organize and manage the load generation is discussed in Chapter 11 – Load Generation in APM Best Practices. You need to be familiar with this chapter if you are going to deliver an audit pre-production. You will want to ramp up over a five minute period and then hold the virtual user population constant for at least 10 minutes (steady-state) and then ramp down over 5 minutes. You will focus your analysis on the steady-state periods.

If you want to try comparing different configuration settings or code releases, use the exact same test plan and execution. Use Introscope Reports to make the precise comparisons, once you have identified the key performance metrics, always comparing those steady state periods.

If you want to evaluate performance characteristics (and really, who doesn't!), you need to allow for a *ladder* of test volume, usually on the order to 10 vusers (virtual user), 50 vusers and 100 vusers. If the client is normally testing to 5,000 vusers, then break this into an appropriate ladder (100, 1000, 5000). This approach allows us to demonstrate that it is absolutely possible to characterize performance with a small, consistent load – something that the load tool vendor very often challenges (they get paid by the vuser).

You can have longer duration tests such as a *soak test* and also stress-to-failure. As long as they are some multiple of the 10 minute steady-state period, you can compare invocations counts by dividing the soak test steady-state time by 10, to obtain the scaling factor, and then dividing the soak test invocations by this factor. This way you can directly compare invocation volumes with the baselines established in QA.

While the client may 'believe' they have consistent load generation, Introscope visibility is often ruthless in establishing how poor the 'consistency' actually is. This is a very important determination for the long term success of using APM in QA, especially when they are focused on comparing the performance gains between release or configuration changes. The variation will often exceed the ability to accurately measure the change and will often present conflicting results. The Application Audit cannot hope to fix this problem – our job is to improve visibility and characterize performance. So if you find that the load is not reproducible, at any level, simply note it, and move on. The performance characterization process is the same. You just cannot have complete confidence in the findings and

recommendations. For this reason alone you must never draw a conclusion about what is broken in the app. It may very well change once the defects in load generation are addressed.

During your scope call, make sure you get answers to the following:

1. Can you generate a manual load against the target application?
2. Do you have automated load generation scripts for the target application? How is load testing invoked?
3. Will we have ready access to load generation for the duration of the audit? Are there any constraints in accessing the environment?
4. Can you generate a ladder of test volumes (vusers) during the testing?
5. Do you ramp-up and ramp-down as part of the test profile?
6. Do you run a soak test? What duration and load volume?
7. Can you load the application to failure?

Baselines

The tool and process for generating and comparing data is called *Baselines*, which is discussed in Chapter 12 - Baselines in APM Best Practices. You need to be familiar with this Chapter in order to deliver an audit successfully. Be careful to ensure that the client is not confusing *Baselining* – the APM functionality that is based on heuristics that are gathered automatically, thresholds applied and generates the "What's Interesting" overview dashboard. What baselining lacks is the ability to specify the interval on which the threshold is calculated. The App Audit version of a *baseline* specifically denotes the intervals which represent the correct testing outcome, as well as the criteria under which the test was conducted.

To mentor folks in delivering the Application Audit, use the presentation **LCMM-1 App Audit Processes.ppt**. This depends on **LCMM-1 Baselines.ppt**. And you may also want to consider additional modules for configuration and managing agent configurations. See the list of Best Practice Modules applicable in the References section.

If you only want to overview the App Audit, in the context of the best practices overall, and maybe at a higher level, then use one of the workshop presentations, either **APM BP – QA Testing and Acceptance Criteria**, for the testing team or **APM BP – Config Tuning – JVM and APM**, for the APM practitioner.

DELIVERY

Access

Before going on-site you need to confirm that you will have someone to access the APM Workstation. It is very rare that you will be able to login to a production system, let alone hook your laptop into the operational network. What you may have had latitude to do during a pilot is simply not allowed with production or other environments.

In addition to APM Workstation access, you will also want to confirm that you will have PowerPoint and a browser with connectivity to the outside. You will need to screenshot and annotate – PowerPoint is best for this. You can also use Word or something equivalent. Screenshot everything! This is all you will have on which to base your analysis and recommendations. You will also be transferring files (ppt and pdfs, typically) which you can do via a browser-based (webmail.ca.com) email session. I find that it is very rare that even a USB port is open on any machine that touches the production network. Be prepared!

Activities

In this section we will discuss how to use you time on-site. With a competent QA organization, you can usually complete testing in about two days, you can then spend another day or two building configuration thresholds, reports and dashboards, and you will want to allow another half day to finalize the report and a short meeting to deliver the findings and recommendations. My preference is always to do this in PowerPoint, which results in the shortest prep time in the reporting phase. PowerPoint also has one special advantage (guideline) – your recommendations should never exceed a single slide! And if you can't read the slide (from a distance) – it's too much! Some folks prefer Word – I guess they feel a lot of pages are 'better'. Certainly, if you have five days or more for the HealthCheck a written report is often the better choice.

Application Architecture Discussion

Allow 20 minutes to review the physical deployment with some details for the platform configurations and types of applications being monitored. You will then use the following questions to quickly assess the current teams "facility" with APM:

- How many agents are deployed for application X?
- How many metrics are you seeing currently for an agent monitoring application X?
- What access restrictions are applied for the APM environment in production?
- Who get alerts from the APM technology?
- How many metrics are managed by the APM –MOM today?

Why are these questions important? The biggest impediment to successful APM is simply having the 'wrong' scope of responsibility. In organizations with weak APM, everyone *assumes* that someone else is "taking care of the environment". If your team can't answer these questions accurately, you can safely assume that *no one* is taking responsibility. You need to help folks understand that this is part of their job! You can't exploit APM until you appreciate what it does and how it works – and make that part of your daily conversation.

Load Generation

The entire success of a QA Application Audit depends on the ability to reproduce a load test. You should be able to run the same test, three times, and get EXACTLY the same results for response time and invocations, for the period that the application is at steady-state. Often, you will not. This is critical to document and understand as it will impact the accuracy of your thresholds as well as any overhead testing that you might attempt as part of the Configuration baseline. There are just so many ways to do testing wrong, that clients have no idea of, until they are revealed by APM visibility.

Your first rounds of tests should be this: three runs 20 minutes each, with either the default instrumentation or whatever they are currently using. Make sure to exercise the application a bit before starting the tests (what I call a 'tickle') and run the three tests in quick succession. Pick a couple of metrics and see how consistent they are over the middle 10 minute interval which should represent steady-state.

Basically, your audit findings are as good as the load reproducibility. Conversely, 'crap' reproducibility == 'crap' audit findings.

If you find that load generation is simply not reproducible, simply going through the audit process is valuable to help the client understand how to use APM technology, other than simple alert monitoring. Just ensure that you manage expectations appropriately. You cannot expect to accurately measure the performance differences between releases when you cannot even reproduce the test! If you don't like the result – simply repeat the test until you get a value that is acceptable! Of course this is shockingly wrong – but this is the reality. If you want the 'truth' – fix the load reproducibility.

Collecting Baselines

The three baselines are fundamental to ensuring that you have a compatible monitoring configuration, appropriate starting JVM and app server configuration, sufficient visibility, and are focusing on the key performance indicators. When a client pushes back and instead simply demands to "measure everything", employing the heaviest monitoring configuration, reporting on every value encountered and deferring all of the analysis to some time in the future – you are going to struggle with this dysfunction until they finally get the point. Monitoring is a balancing act between visibility and analysis. When we know what to look for, it is easy to do the analysis. But when the client really doesn't know what to look for, they will instead gather everything possible and set it aside, intending to use it later. It is some form of IT procrastination! In fact, the data volumes are so large that they never successfully get to use it. The APM best practice instead tries to move the focus on immediately assessing the visibility obtained and immediately conducting the initial analysis to generate a baseline. This results in a tool to help accelerate baseline, effectively reducing the volume of data to manage. Before APM BP – we have 5000 metrics, relationships unknown. After APM BP – we have a baseline with 10-20 metrics, relationships fully exposed, candidate metrics for thresholds, and a report template (the baseline). We don't need to manage all of the data because we now know "what matters" for a given application.

Ultimately, in order to become proactive, we need to do all the work pre-production and not wait for the production incident to start analyzing the data. The Application Audit is the first process step to becoming truly proactive.

CONFIGURATION

You can begin to evaluate the configuration baseline as soon as the application is functional, usually during *Functional Testing* where the application is being exercised manually, with only a few users. If you have not already identified a suitable *'gold' configuration* for the candidate application, you should make a quick evaluation of what APM configuration options will be compatible or useful. If you are not allowed to do this, then restrict your work to the default-typical configuration only. Do not allow testing to begin with the heaviest configuration possible – you might get lucky but you will almost certainly suffer!

- Monitoring Configuration Compatibility
  - Starting with the agent deployed but autoprobe disabled, exercise the application for 5-10 minutes.

- Then enable autoprobe with default-typical, restart the app server and exercise the application for 5-10 minutes.
- If appropriate, add in any resource powerpacks (MQ, CICS, etc.), restart and exercise as before.
- If appropriate, add any app server powerpack, restart and exercise as before.
- Now move to default-full, restart and exercise as before.
- If there are any CMTs, enable these, restart and exercise as before.
- Finally, add in any portal and webservice functionality, restart and exercise as before.

To analyze the results, look first to the number of metrics and note how many metrics each configuration generated. Grab a screen shot of the Metrics TypeView, for each configuration to confirm that the configuration is useful. If it doesn't generate any metrics, you really should not use that configuration even as it has no effect on the agent. Now use the Search Tab to find the highest response time metric. View this metric over the testing period and assess if the response time was affected with each configuration change. Recall that every Java component has a very poor initial response time the first time it is invoked. If a component was only exercised once then this has no meaning. If it was exercised five or ten times, discard the first value and consider what remains.

If the application "blew up" or otherwise failed to operate as expected, do not use that configuration again – it is not compatible. Whatever compatible configuration remains, that is now the candidate 'gold' configuration for this type of application, which further validated. All other instances of similar application can be assumed to be compatible so that it is not necessary to undertake a configuration baseline for each and every application receiving monitoring. However, any applications that have a different functionality and/or resources should have a configuration baseline evaluation.

Here is a list of different types of applications where one successful monitoring configuration can be assumed to be good for other members of that group:

Common ApplicationTypes
Transaction
Servlet/JSP
Directory
Database
Messaging
POJO
.Net
EJB
Portal
WebService

Detailed Application Types
BATCH
SEMI-BATCH
Scriping Language (Perl, etc.)
WEB-INTERACTIVE
WEB-EAI
WEB-SOA (web service)
WEB_CLOUD
LEGACY-CLIENT-SERVER (please note)
MESSAGING-MIDDLEWARE
TRANSACTIONAL-MIDDLEWARE
EVENT-MIDDLEWARE (note vendor)
B2B_EDI
B2B_SWIFT
B2B_HIPAA
B2B_ROSETTANET
WORKFLOW
WORKFLOW-HIGH-VALUE
PACKAGED-APPLICATION (note vendor)
PORTAL-IN-HOUSE
PORTAL-WEBSPHERE
PORTAL-WEBLOGIC
PORTAL-OTHER (describe in notes)

TABLE 2: SAMPLE APPLICATION TYPES – SIMPLE AND SPECIFIC

These lists are arbitrary definitions but will help you categorize the variety of applications you will encounter. Configuration baselines are a little work but the results can be leverage for many applications, when you have a consistent set of categories. Feel free to define other categories as needed but then be sure to update them in your sizing and survey tools as you will want to have a single set of categories. Look to the *Phased Deployment Model* to understand how a 'gold' configuration strategy can accelerate large-scale deployments.

- Memory Utilization

The next analysis is to consider the GC profile: Bytes in Use and Heap Size. The first time an application is run, it very often progresses with default JVM and Application Server configuration parameters. This will be evident, even with a short, manual exercise of the application. If you see big swings in the Bytes in Use, or frequent increase in the Heap, it is likely that no tuning of the JVM and server

have been implemented. Making this tuning is beyond our audit scope but you will want to notify the developers to make a correction prior to beginning load testing. Otherwise, the load testing will be a waste or not be comparable to prior baselines.

A good JVM configuration has attributes of smooth Bytes in Use without massive GC activity. Memory use should not immediately move to the maximum, considering we are only doing manual load with a few users.

- App Server Configuration

If PMI or JMX metrics are available, the App Server configuration will usually be reflected in these metrics. The manual load is usually insufficient to reveal any shortcomings in these settings but confirming that they are being collected, and what values are present, is enough at this stage. Clearly, you cannot be expected to tune the application server configuration if the metrics are not being collected. Even as many of these metrics are static, and can be found in the startup script and other configuration files, we may want to make the values part of the baseline report. This lets us quickly compare if the settings are consistent, or otherwise changed, where comparing different versions of the application.

APPLICATION

With the monitoring configuration determined to be compatible and hopefully any issues about the app server and JVM configuration addressed, you may now let the QA team test the application as they would do normally. You will need a reasonable test of the various use cases in order to confirm sufficient visibility to understand performance problems. *Transaction Trace* is the tool to use as it clearly indicates the call stack, for each of the use cases, and allows you to visually confirm that you are seeing enough of the backend interactions (60% is the minimum goal) and that you have at least 3-8 levels of stack. Transaction Trace will also reveal any number of poor implementation patterns that can result in scalability problems.

Here are a couple Transaction Traces graded Good, Needs Help and Bad:

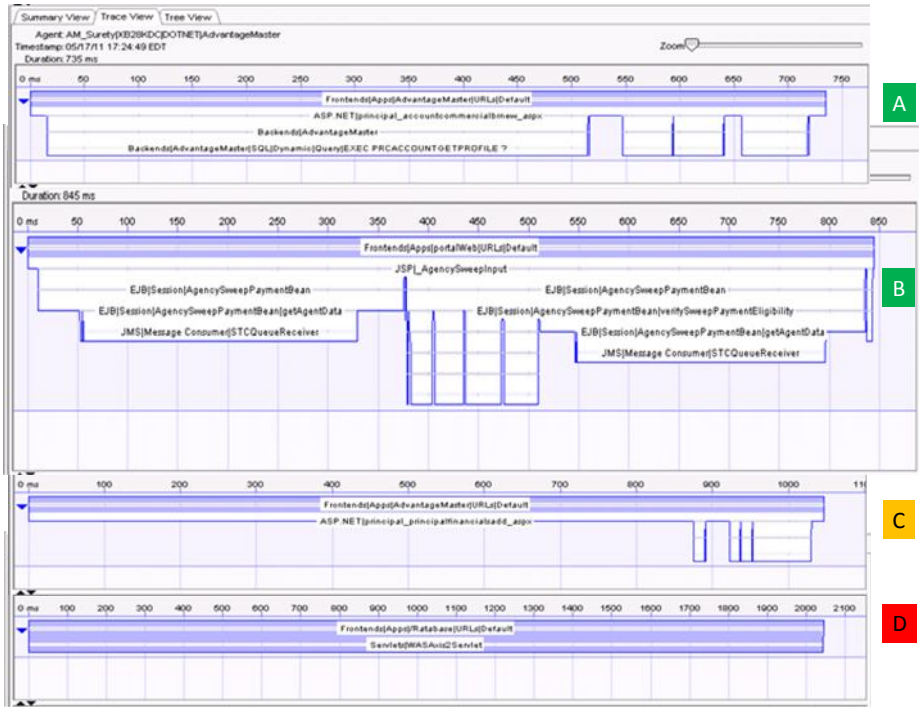


FIGURE 3: GOOD, ACCEPTABLE AND POOR EXAMPLES OF TRANSACTION TRACES

A: This trace is quite good. We are concerned with the bottom of the trace and ensuring that we have at least 60% of the exits, from the application server, covered with instrumentation. If you 'click' the lower components there will be a report pane below with you tell exactly what percentage of the overall transaction is covered by that component. Just add up the lower components and ensure you are 60% or greater.

- B: This trace is also good. The bottom of the trace is not at the same consistent level, which is due to the mix of components, MQ and SQL, that make up the 'exits'. Less of the MQ activity may be traced but there is clearly enough for us to be able to 'blame' the MQ component in the event the response time was excessive or an error trace was indicated.
- C: This trace is poor and while it might be enough to do testing and maybe even go to production, the visibility gaps are evident. Maybe 10% of the exits are covered. If we have a lot of transaction like this one, in terms of overall visibility, then we really need to

have a conversation with the developers to understand what frameworks and components we might be missing. We would then use CMTs (Custom Method Tracers) to correct the monitoring configuration and fill in those gaps.

- D: This trace is unacceptable. We only have the entry servlet and no details as to the components that comprise the 'exit'. Since we cannot see what resources are involved, when this transaction's response time degrades, we will not be able to triage which resources is responsible.

As you load testing is ongoing, there will be periodic sampling of the transactions to help you build up a 'zoo' of interesting traces. You will then go through the traces to see what kind of transactions you are experiencing. Grab a screenshot of each new 'animal' you encounter. You will often find the same pattern for the same transactions. After you have surveyed what transactions are available, go back and 'grade' each one.

Summarize your findings as follows:

- What are the characteristics of the monitoring configuration used?
    - What types of extensions
    - How many metrics
    - Screenshot of the Metric Count Typeview
- How many transactions have sufficient visibility?
- How many are 'passable' but would benefits from additional work to enhance the visibility?
- How many transactions have insufficient visibility and require additional monitoring configuration (PowerPacks, CMTs, etc.) to resolve.
- Make an estimate for the proportion of each situation as a percentage of the total transactions observed.
- Which transactions reveal poor design patterns (death by 100 cuts, etc.)

PERFORMANCE

For the performance baseline we must have consistent load. Manual exercise will usually not be enough to determine a performance signature and has no chance of every being reproducible. We also do not require a high-volume load test – a *ladder* of tests in 10, 50, 100 will often be enough to show performance trends. Every app will have a different goal. Where possible, try to use the exact testing regime. Move next to a ladder approach, if the QA team is agreeable and, of course, if the testing schedule allows. If full production-load testing is done, either as a *soak test* or *stress-to—failure*, this will also be useful, especially if we want to forecast capacity for the application.

Selecting Metrics for APC: Availability, Performance and Capacity

Finding candidate metrics to determine KPIs (Key performance Indicators) is straightforward, provided that you have reasonable load generation and a plan. First, focus on the metrics for APC. Then consider other metrics, such as concurrency and stalls, especially if performance problems are evident. Continue with additional metric categories as the candidate application has them available or as seems appropriate for situation at hand.

The most important lesson for the client is to appreciate that you do not need to report on everything that the monitoring configuration has to offer. You simply need to learn how to find out which metrics matter. You need to start reducing the data stream to the things that will help you make decisions about performance. Ultimately it is not the total number of metrics – more does not necessarily mean better. What matters is that you have the 'right' metrics. These come from consistent testing, and having information around the performance anomaly – not just what is happening after the incident has occurred.

AVAILABILITY

No matter what, always build two metric groups for availability. These will use the Connected and Metric Count metrics. Threshold each and then create a summary alert called Summary_Availability, and associate with the two availability metric groups with that alert. These steps will serve as the template for the more complicated performance and capacity metrics that will be selected shortly. Availability metrics are less complicated because there are only two needed, they are obvious as to what the represent, they are application specific and they absolutely work every time. They even work with the Console Lens so you don't even have to build custom dashboards for many applications – just use the sample dashboards. For many organizations this Summary_Availability alert is the only alert that need to be integrated with the Trouble Management system.

PERFORMANCE

For performance metrics (and later capacity metrics) we really need to use the Introscope Workstation to help us find the best candidates. Hopefully, the testing is running concurrent with your analysis. This is more critical for the configuration and application baselines, as the

traces will age out quickly and you'll want to make sure the various monitoring configurations are correctly established prior to manual testing.

Using the best visibility configuration, and before the Tier 1 data ages out (7-14 days typically), you want to do the performance baselines. Older data works just fine but there is nothing like the nuance that 15 second resolution can provide. Since we are only looking for candidates, and not directly comparing release A to release B, we don't need to be fussy about the exact period of the test. Use your discretion but if the testing occurred yesterday, then setting the historical period to last 24 hours will be sufficient.

Next, click on the candidate agent (comprising the candidate application), lock to 15 second resolution, select the search tab, search on "average", and then toggle the Value Column to sort the values largest to smallest. Here is an example:

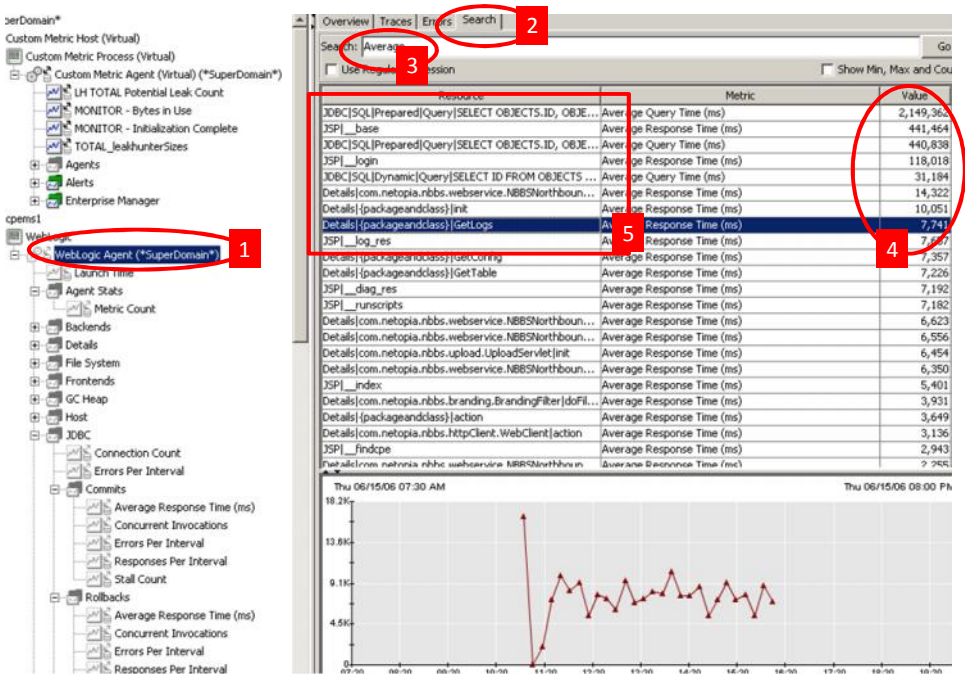


FIGURE 4: SEARCH FOR PERFORMANCE CANDIDATE METRICS

Whatever shows up as the top 10-20 resources, this is your focus. Begin by <SHIFT><CLICK> the top components you want to consider. This will cause a graph of those components to be displayed. Where the curves are proportional, the largest value is the candidate. If it is too close to tell, then pick a few. Depending on the top of load profile being used, steady-state or ramp , the response times will have different characteristics and utility. Here is a table to consider the differences:

Load Profile	KPI - Performance	Heart Beat
Manual	Biggest	n/a
Steady-state	Largest and most consistent value	n/a
Ramp	Degrades with load (increases). Could also be bottleneck.	Consistent values independent of load.

TABLE 5: CLASSIFYING PERFORMANCE METRICS AND LOAD

Manual load offers the weakest criteria – simply pick the biggest values, unless something more consistent is apparent.

With Steady-state load, you are looking for both the largest and the most consistent. But since we are at steady-state we cannot distinguish any *heart beat* metrics. Heat beat metrics are important because they are another type of availability metric. Any degradation

in a heart beat metric will let us know the application is failing, before it is reflected in a decrease in the Metric Count or Connected metrics. This will make our availability alert even more reliable – once we have validated that the heart beat metric is accurate for availability.

With Ramp load, you are looking for those metrics that degrade (increase) with load and we will generally take the largest but you must be careful not to select a bottleneck metric. This is not immediately obvious because you need to first reflect on the application baseline to confirm that this metric is participating in a critical transaction and not responsible for the overall response time for the transaction (bottleneck). Those metrics that are independent of load are the heart beat metrics.

If you select a Bottle-neck metric for your KPI it is not the end of the world. You do want to understand where your bottlenecks are but given that they are going to get attention, that means they are likely going to change – and that means a little more work, later on, to effect the change. We will validate the utility of these metrics when we validate the dashboard. And we also have some criteria to consider before we confirm that the metric is stable enough to justify a threshold.

CAPACITY

For capacity metrics, the analysis proceeds the same as for performance metrics, with the exception that we are now searching on "Responses" instead of "Average". This gives use all metrics of the Responses per Interval type, for the agent under evaluation. Here is an example:

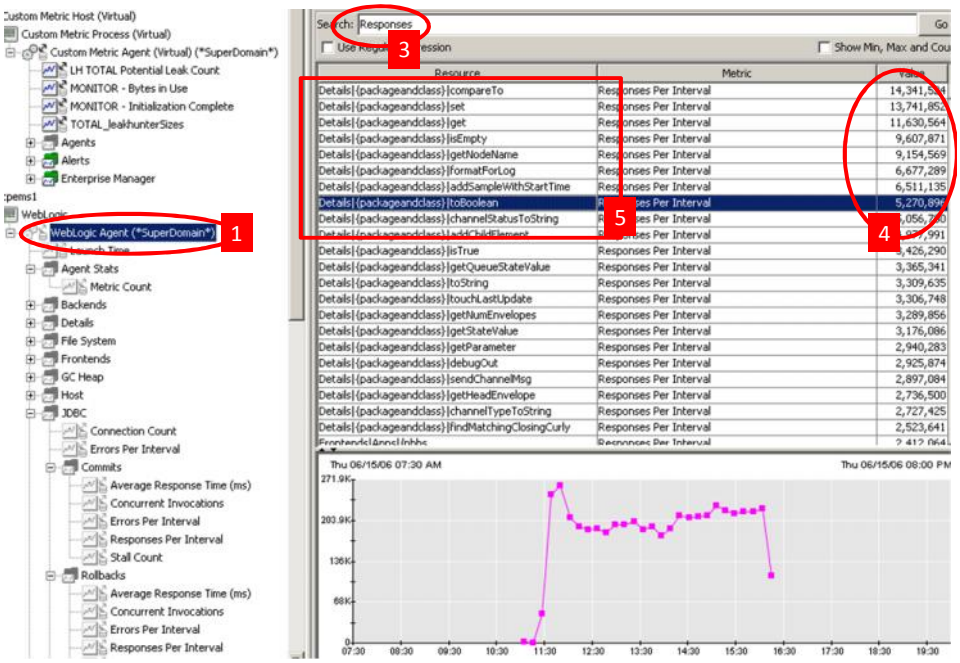


FIGURE 6: SEARCH FOR CAPACITY CANDIDATE METRICS

*** Note – step 2 is missing because the screenshot did not include the Tabs ***

Whatever shows up as the top 10-20 resources, this is your focus. Begin by <SHIFT><CLICK> the top components you want to consider. This will cause a graph of those components to be displayed. Where the curves are proportional, the largest value is the candidate. If it is too close to tell, then pick a few. Depending on the top of load profile being used, manual, steady-state or ramp , the response times will have different characteristics and utility. Here is a table to consider the differences:

Load Profile	KPI - Capacity	Heart Beat
Manual	Biggest	n/a
Steady-state	Largest and most consistent value	n/a

Ramp	Degrades with load (decreases). Could also be bottleneck.	Consistent values independent of load.

TABLE 7: CLASSIFYING CAPACITY METRICS AND LOAD

Manual load offers the weakest criteria – simply pick the biggest values, unless something more consistent is apparent.

With Steady-state load, you are looking for both the largest and the most consistent. But since we are at steady-state we cannot distinguish any *heart beat* metrics. Heat beat metrics are important because they are another type of availability metric. Any degradation in a heart beat metric will let us know the application is failing, before it is reflected in a decrease in the Metric Count or Connected metrics. This will make our availability alert even more reliable – once we have validated that the heart beat metric is accurate for availability.

With Ramp load, you are looking for those metrics that degrade (decrease) with load and we will generally take the largest but you must be careful not to select a bottleneck metric. This is not immediately obvious because you need to first reflect on the application baseline to confirm that this metric is participating in a critical transaction and not responsible for the overall response time for the transaction (bottleneck). Those metrics that are independent of load are the heart beat metrics.

If you select a Bottle-neck metric for your KPI it is not the end of the world. You do want to understand where your bottlenecks are but given that they are going to get attention, that means they are likely going to change – and that means a little more work, later on, to effect the change. We will validate the utility of these metrics when we validate the dashboard. And we also have some criteria to consider before we confirm that the metric is stable enough to justify a threshold.

Now you should also notice that most of the metrics in this screenshot have values that violate our general guideline of not exceeding 5,000 invocations (Responses per Interval) in 15 seconds. Someone dropped in some CMTs that they thought were interesting, probably using the dreaded "com.myapp.*" to thoughtlessly select their methods for receiving instrumentation. This will result in excessive overhead and waste of APM capacity. As easy as it is to make this mistake, it is even easier to detect it and then turn it off. And now that you have found this abuse of APM… you get to rerun your testing because the response times are not reflective of a more 'correct' monitoring configuration (low overhead). It makes a big difference. When you rerun the test - you will get to show exactly how much of a difference. All fun stuff… but not reflective of a performance signature for the application.

- GC/Heap

In general, a graph of the Bytes in Use metric is all that is required to capture the state of memory management. This becomes a little more interesting when there are multiple instances to consider. Building a metric grouping is then indicated. Generating a threshold, say for when the heap is at 80% - this requires a Management Module Calculator. These can be fun to build but in reality, there are better metrics to indicate that the application is doomed, which you will find in the response time, invocation or heartbeat metrics. Some folks really hold on to the % heap available… and all that noise – simply because that is what they know and are used to. When these folks start to realize the detail we have before the heap is exhausted… this is where they become converted. Folks just never imagined being able to correlate performance and memory utilization. Sometimes even to the error that triggers the memory mismanagement.

Where your monitoring configuration includes CMTs for Instance Counts, you will want to do a Search on those metrics and see which ones are growing or balanced. A *growing* count is a strong suspect because it may not be managed. A *balanced* count will grow and shrink, indicating that it is being actively managed.

- EJB, Servlet, JSP

Not all applications utilize EJBs so you will want to have a separate metric group to present those metrics. EJBs may represent specific transaction use cases so these can often be useful to track version changes.

Servlets are also useful but to a lesser degree. Because they are a common entry point, multiple use cases will be lumped together. The servlet is usually helpful to track the overall volume going through the application (capacity). Look to the JSPs, for correlation with specific use cases. These will usually be much more numerous that the servlets.

- Concurrency

Concurrency; the number of component instances at any time, is fundamental to achieving a scalable architecture. When a component fails to scale via concurrency, this often results in a bottleneck. This can be due to a legacy code that is unaware or unable to function concurrently, or when there are not enough resources to support the growing number of instances. The Search tab will quickly reveal if concurrency is being employed and to what degree. Most metrics end up with a concurrency of zero so it is very important to use the

Search tab technique to build a metric group of the actual components that are concurrent-capable. This is not something you will alert on but there is often a strong correlation when response times degrade and a plateau in concurrency.

- Stalls

Stalls are counted whenever a component exceeds 30 seconds response time. This makes them useful for identifying potential bottlenecks and are often the first metric considered for *acceptance criteria* – establishing KPIs that confirm an application has acceptable performance characteristics and should continue on to production deployment. You will see some stalls when an application first comes up but otherwise, when they appear it suggests the application quality may be compromised. For web facing applications, anything that takes 30 seconds or more results in a poor user experience. A stall metric is not present until at least one stall has been experienced. Again, the Search tab approach will help you narrow a long list of 'zeros' and let you quickly build a metric group of the troublemakers.

- Application Resources

Because every application is different you have to make allowances for the variety of resources that you will encounter. The goal of the audit is to take a look at everything and, where the metrics are significant, assemble those metrics into a baseline to simplify ongoing performance evaluation. Initially you will have a broad list of interesting metrics with highly variable response times, which makes them unsuitable for thresholds. Initially you will prepare a metric group for these different categories but it will be unsuitable for dashboards because of the large number of metrics. So you will report on them… until the novelty wears off. Anything significant will have already been exposed during the search for response times and invocations. But some folks will continue to insist on having 80 pages of SQL statements, for a time.

  - SQL Statements

There is tremendous variation in response time and invocations with SQL but they can have some utility when they indicate a specific use case. This is helpful when you do not have sufficient visibility to collect complete transaction traces but otherwise want to know which use case was active when correlating a suspect. From the capacity perspective you only need the largest volume SQL to establish a good threshold. From a performance perspective the presence of extremely long and extremely short response time SQL suggests that an optimal JVM tuning is not possible and that the application will always have unstable performance. This continue until you separate the long duration SQL, from the short, either by moving them to a separate JVM (application partitioning) or using an asynchronous design pattern for the 'workflow' transactions.

  - JMS (messaging)

JMS components will usually be magnitudes smaller that SQL and often used to bolt legacy functionality together. These attributes make better candidates for thresholds and quality tracking. Response times will vary considerably, even as they should always be less than 1 second. For capacity thresholds, JMS is usually reliable.

  - Web Services

Web services can be a metric nightmare, depending on the granularity. The candidate application will employ every level of functionality, from high-level business services, down to low-level format converters. So the impact on performance will be challenging to correlate and document. The same Search principles apply and you may find opportunities to use separate metric groupings for the various levels of web services that you encounter. Not all of these will be useful for reporting purposes. The real benefit will be during triage where you can quickly check through the various metric groups to find the collection that best correlates with the performance problem. Once that correlation is established it will be useful to set thresholds to capture the trouble makers. Load testing alone is usually not sufficient to gather performance signatures as the external services are often 'stubbed' (sham functionality that accepts the call and immediately returns) or not be utilized at realistic loads.

- Where are my CPU metrics?

Measuring CPU consumption can sometimes be point of contention. When your client has been insisting on CPU metrics, it is often because these are what they have always used or are otherwise familiar with. All the other available metrics are not really being considered. This is a challenge because CPU is a measure of consumption, not performance. Response time and volume are the real performance indicators. If the organization is going to manage performance, they need to use the right metrics!

When mainframes walked the earth, CPU consumption is how they were accounted for, in terms of cost and efficiency. That is still the case today where mainframes are involved. And it makes complete sense when you are considering batch computing operations or transactions that are otherwise very short acting and with minimal business process.

With the rise in distributed computing, the emphasis on CPU is vastly diminished in favor of response time and volume metrics because these attributes are now directly measureable on a per transaction basis. We also allow arbitrary complexity in the business transaction because most of what we call distributed computing is really an integration layer of a variety of established systems. This makes the

business transactions pretty much unpredictable, in terms of response times. Tracking these transactions, end-to-end, is the real motivation for moving to APM anyway.

A lot of the justification to measure and track CPU consumption can also be attributed to the rich legacy of capacity planning that mainframe-centric organizations have developed. But this is based on analysis models of 2-tier client-server computing, which simply doesn't exist today – especially where distributed computing is the norm. It's called "n-tier" for a reason. Two-tier models simply don't apply. In fact the complexity of building analytic and simulation models for n-tier is so difficult, especially for validation and comparison with the real world, you have no other choice than to rely on direct measurement of the business critical transactions. This is what APM is doing.

Of course we have CPU metrics. We have the JVM perspective and the OS platform (via EPAgent). We even have extensions to measure the fraction of CPU that a single method consumes that may be used in special circumstances. What we don't have is any significant practice that uses CPU metrics to get visibility into performance problems. Or forecast capacity. Or do anything of significance. No one does. It is not something you do with APM. It is the only thing you can do before APM.

As more organizations learn how to establish performance and capacity forecasting based on APM techniques (direct measurements) it is likely that a new generation of capacity management tools and models, those exploiting the variety of metrics that APM manages, will be established. Today that is simply a dream.

(optional) Forecasting Capacity

When a client has a mature testing practice you will be will be able to stress-to-failure. For some applications this will be difficult because they gracefully handle extreme loads. This is rare. More often, the candidate application is heavily clustered. In this case, the solution is easy: deactivate 80% of the cluster and hit it will the full load. If you really want to understand performance potential, there is nothing more illuminating then watching an application taken to its knees.

If the break point is known, in terms of load volume, ramp up to this volume over a 2 hour period (minimum). Extend the test at least 20 minutes beyond the failure point. This will insure that you have plenty of detail on which to correlate behaviors. The ramp will allow be reflected in key components so that it will be easy to identify when the application fails to scale, what we call the *performance knee*. Once this point is identified you will want to rerun you baseline analysis to identify the key components during the period where performance is no longer scaling, and compare to your 'normal' performance baseline. Any changes may help you further refine which components to alert on.

Here is an example of what you may find:

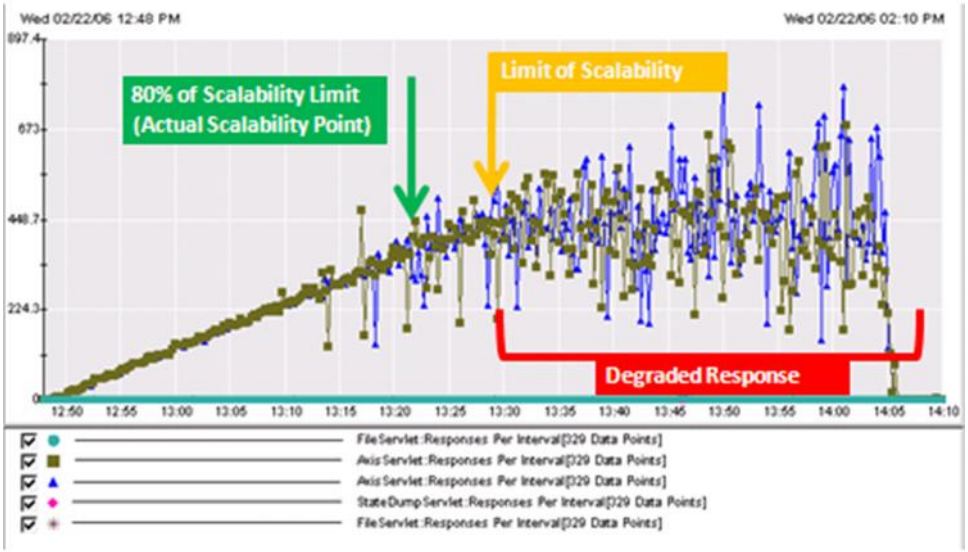


FIGURE 8: FINDING THE ACTUAL SCALABILITY POINT

We are looking at the invocations (responses per interval) for the key components. When the response times become degraded, the application will pause and surge as cache behavior dominates. This is directly evidenced by swings in the invocations. Where this behavior is established is effectively the *limit of scalability* – the application may survive but it is not longer scaling with load. Coming back from this point to 80% of the invocation volume is where you would establish the *actual scalability* for the application, after which, the application is no longer going to scale 'gracefully'. At this 80% point you will identify the key metrics that best represent the 'caution' alert. While the 100% point (limit of scalability), would be a good point to set the 'danger' alert threshold. Of course, your project plan may want to confirm a *target scalability* (performance goal). When the *actual scalability point* is greater or equal to the *target* scalability – you will have met with success.

(optional) Auditing in Production

A production audit is useful when there is no QA practice available, and we want to collect the performance signature, or if the production signature is being collected in order to improve the QA practice. Understanding the differences between production and QA can be used to improve or correct QA test plans, which is absolutely critical to

Unless your client has a very mature monitoring practice, consisting of, at minimum, a long-term (> 6 months) APM implementation and validated visibility , there is no way to avoid waiting multiple weeks for consistent data on which to develop an accurate baseline. For example, here is an application with terrible performance and poor consistency.

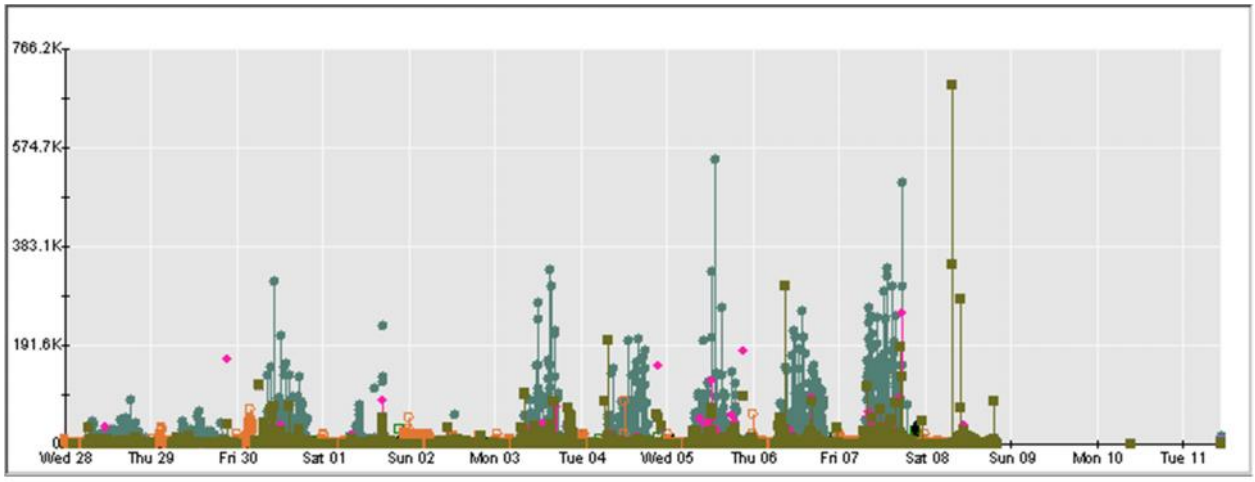


FIGURE 9: TERRIBLE PERFORMANCE AND POOR RESPONSE TIME CONSISTENCY

These response times are just completely wrong (compare measured with typical). You must first address the performance problems, repair them and confirm the situation is improved before an application audit is appropriate. What they need is a *firefight* or *performance tuning* engagement. Both situations are predicated at having resources available to make corrections to quickly bring the application performance under control. An audit does not make that requirement.

Here is an example of a reasonably consistent period for baseline analysis:

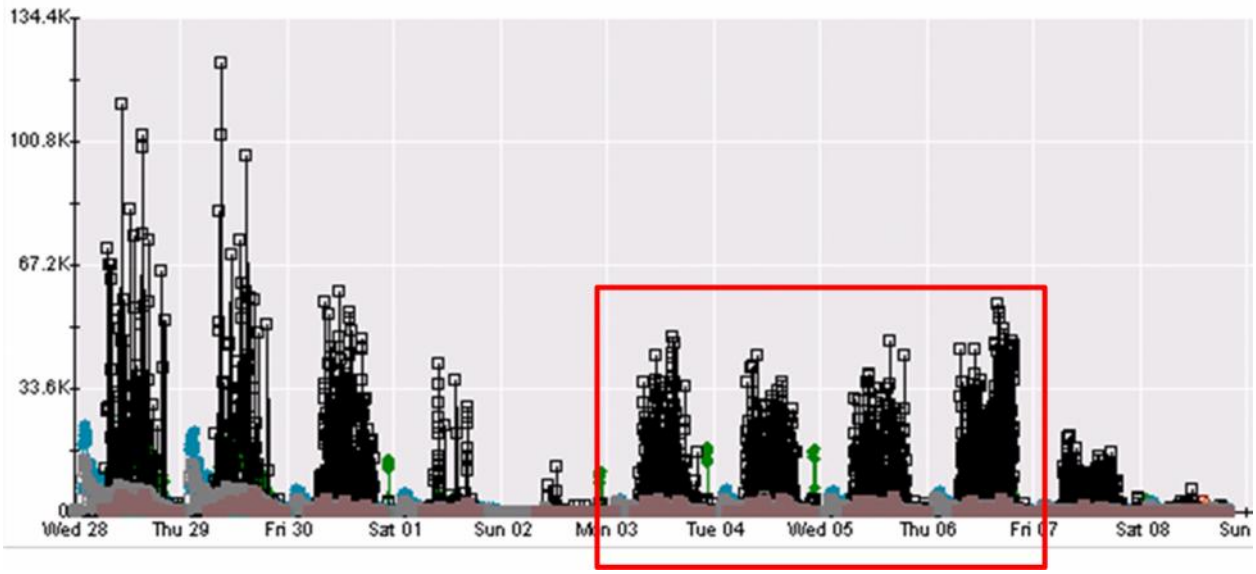


FIGURE 10: REASONABLY CONSISTENT INVOCATION VOLUME FOR BASELINE ANALYSIS

For this component we can see that the invocations have had some consistency and we will have at least three samples on which to conduct the analysis. We should also be concerned that the prior week is something quite different! Time-of-day or Time-of year considerations may play very heavily into whether or not a production audit is even feasible.

While this would be acceptable reproducibility for production, this would not be considered reproducible during QA. In QA we expect the load to be exactly reproducible – it's synthetic! We should have total control.

Defining Thresholds and Alerts

For many applications, especially when you have stakeholder participation, it will be relatively easy to select appropriate metrics that best represent availability, performance and capacity. But if you do not have the experience of the client to draw on, to help you select the best metrics for thresholding, then you will need to select them on your own.

There are actually four categories of metrics that you should implement metric groupings for: Connectivity, Availability, Performance and Capacity. And there are two major techniques to thresholding the metrics groups. You will also plan for summary alerts for each of these categories but as more of a template.

The challenge with thresholding will be in validating that the thresholds are correct. If not, you will generate too many alerts and thus frustrate the operations team.

You also need to plan for two consumers of alerts. The operations team in the obvious recipient for availability alerts but you do not want to automatically send the performance and capacity alerts because operations really only has too mechanisms to respond to an alert. They can either restart the affect system, or they can open a bridge… and then restart the system. As many applications are clustered, in order to improve availability, allowing operations to restart one sever means that that server's load is then distributed across the remaining servers. When the alert if performance or capacity related, this means that the restart can actually cause a cascade failure of the remaining servers – and the entire service will be lost. Thus you need to plan for an alternate channel to advise the application stakeholders that a performance or capacity problem is occurring. This may or may not result in restart of the service – but the app team needs to make that judgment.

It is also a reality that you may not have metrics on which to base performance or capacity decisions that the application stakeholders are comfortable with. So the absolute minimum that you can expect to implement is an Availability alert defined as a Summary alert on the metrics for Connectivity and Metric Count.

When you have a large number of candidate metrics for Performance and Capacity, you can narrow the selection by considering the max and max-max values and how they relate to the average value being experienced. When you select a metric for threshold you want to ensure that the metric does not vary wildly around its average value. If it does, then this metric will always be exceeding the threshold and often this will occur when the application is otherwise not in trouble – and confidence in the monitoring will be diminished. The max and max-max values really give you a sense of how stable the metric is. The more stable – the better candidate for a threshold value.

The following table can be used to help exclude candidate metrics from consideration for thresholds as well as score the overall deployment risk:

Analysis Step	Most Metrics Meeting Criteria	Few Metrics Meeting Criteria	No Metrics Meeting Criteria
Average value Is the app performance consistent with similar apps?	False •Servlet is a concern •EJBs a slight concern	**True** •JMS Publisher – quite good •SQL is reasonable	False
Max value – rule of 10 Are consistent thresholds possible?	False •JDBC unacceptable •EJB unacceptable	**True**	False
MaxMax value – rule of 100 Is the performance predictable?	False •EJBs •JDBC-update unacceptable •JTA unacceptable	False •Servlet would be a candidate once performance is improved	**True**
Subjective Conclusion Is production deployment advised? (# tests == TRUE)	0/3–Suspect 1/3–Weak 2/3–Stable 3/3–Solid	0/3–Problematic 1/3–Significant Risk 2/3–Moderate risk 3/3–Low risk	0/3–Inconclusive 1/3–Tuning likely 2/3 –Tuning needed 3/3–Unacceptable

FIGURE 11: SAMPLE SUBJECTIVE CONCLUSION MATRIX FOR AUDIT RECOMMENDATION

The Average, Max and Max-Max values are obtained with the Report Generation using tables for the various metric groups. See section *Delivery – Activities – Preparing a Baseline* Report for the example on which this analysis is based.

First, you will review the baseline report considering only the average values, asking the question "Is the performance of this component consistent with other applications that have the same component? Simple count the number of metrics that have acceptable response times and indicate which test is then true: Most are acceptable? Few are acceptable? None are acceptable.

Second, consider the max value and see which of the components have a max value that is less than 10x the average value. Metrics that pass this test are good candidates for thresholds. The quantity of metrics that pass this test are then considered as before: Most are suitable? Few are suitable? None are suitable?

Third, consider the max-max value and see which components have a max-max value that is less than 100x the average value. Metrics that fail this test are concerns due to the excessive variation that they present. They cannot be used for thresholds and will cause unpredictable performance problems. The quantity of metrics that pass this test are then considered as before: Most are suitable? Few are suitable? None are suitable?

The Subjective Conclusion is then a result of the Total "TRUE" in each column, which is looked up at the bottom of each column. Thus this application's baseline analysis results in "Suspect – Moderate Risk – Tuning Likely" – be afraid! The only suitable metrics for threshold would be the JMS: Queue Sender and JMS: Topic Publisher.

This table also brings up the topic of *acceptance criteria* – how do I know that the application has sufficient acceptable behavior to be deployed to production? Many clients are more focused on the meeting the deployment schedule – software quality is a much lesser priority. While a single audit will not change their behavior you can still point out that "based on the variation in the metrics" that you would have concerns deploying the application to production. As unstable apps frequently have bad experiences operationally, at least the client cannot come back at you for failing to point that out in advance!

This brings up the reality of "organizational maturity" with respect to acceptance criteria and making software quality a priority over the schedule. You can't really address that problem with an audit process alone but the 'snapshot' that the audit represents is really putting a stake in the ground. In the future, when the client has had multiple audits predicting multiple production problems, they will then have confidence in the Application Audit to identify acceptance criteria and thus prevent problematic apps from getting deployed at all.

DEFINING THRESHOLDS

Once you have completed some load testing, use the search tab to identify the largest response time metrics, choose the best representative value and configure the threshold values. Here is an example of where to put thresholds for the *Metrics Count* metric:

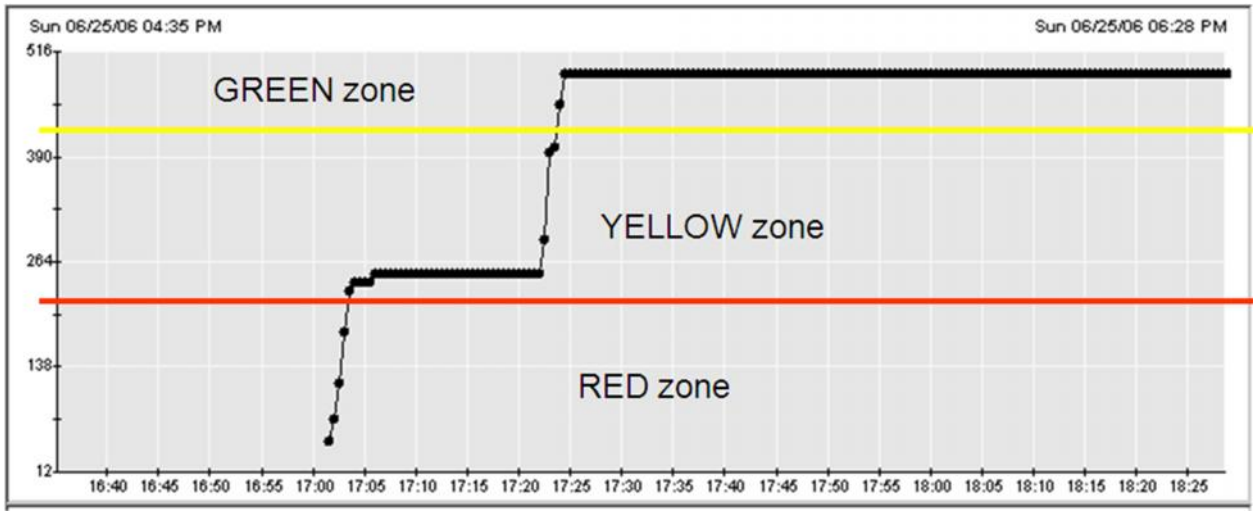
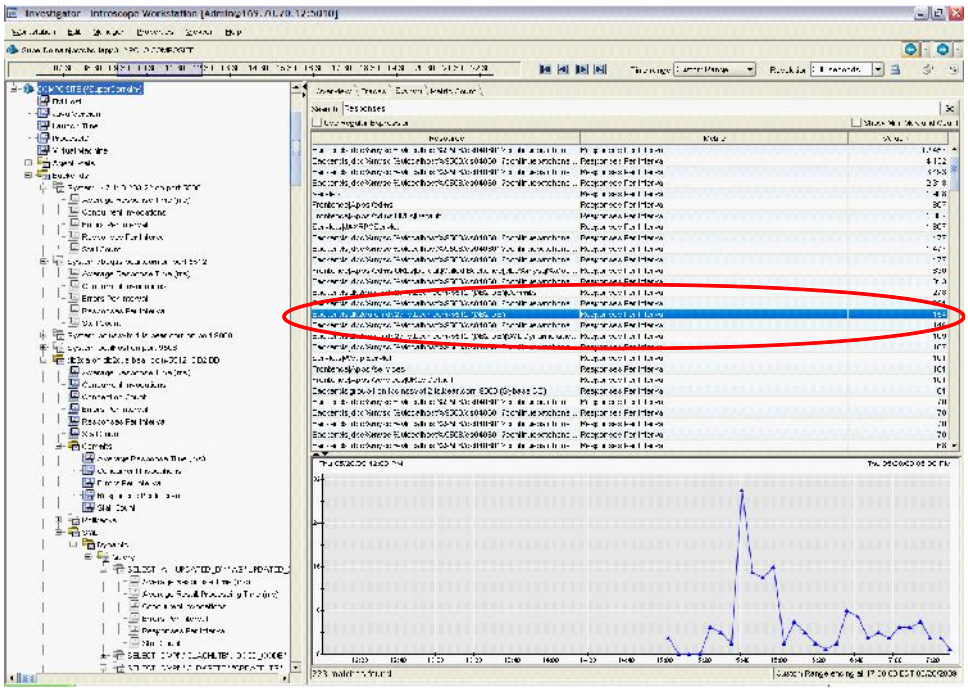


FIGURE 12: EXAMPLE THRESHOLD: METRIC COUNT

Establishing the metric groups takes of bit of multi-window coordination in order to have maximum productivity. First open separate Investigator and Management Module windows. Define the metric groups and create a number of slots to receive metrics via drag-n-drop. Then switch to the Investigator window and use the search tab to find the metrics of interest. Verify that you have appropriate metrics displayed in the Search pane.

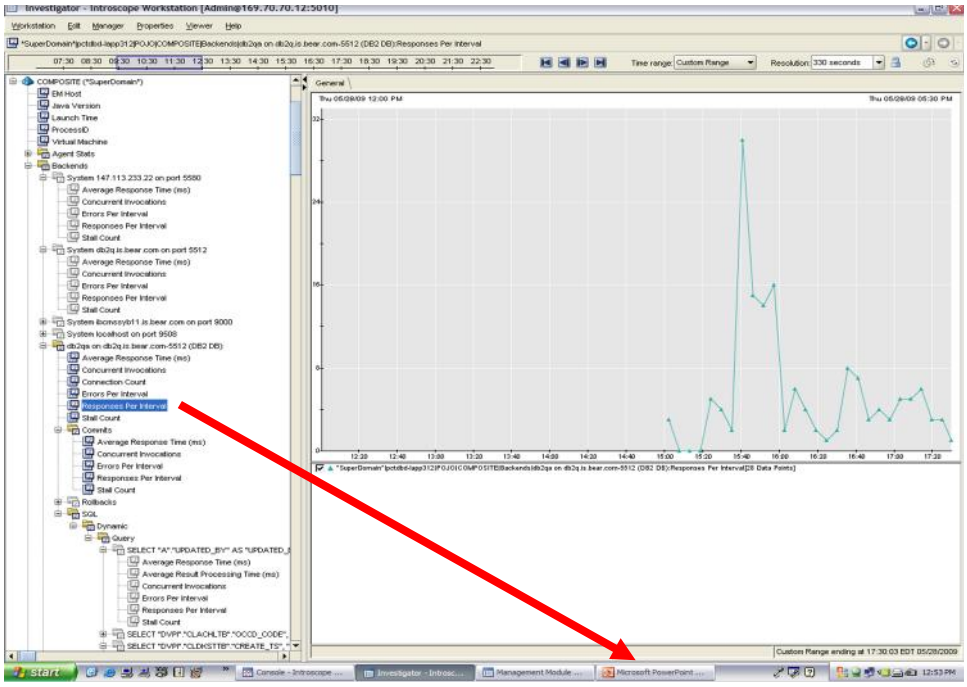
For each metric of interest, double-click, which will then take you to the location of the metric in the Investigator Tree (left pane).



Identify a metric of interest, double-click ---

FIGURE 12: BUILDING A METRIC GROUP – METRIC OF INTEREST

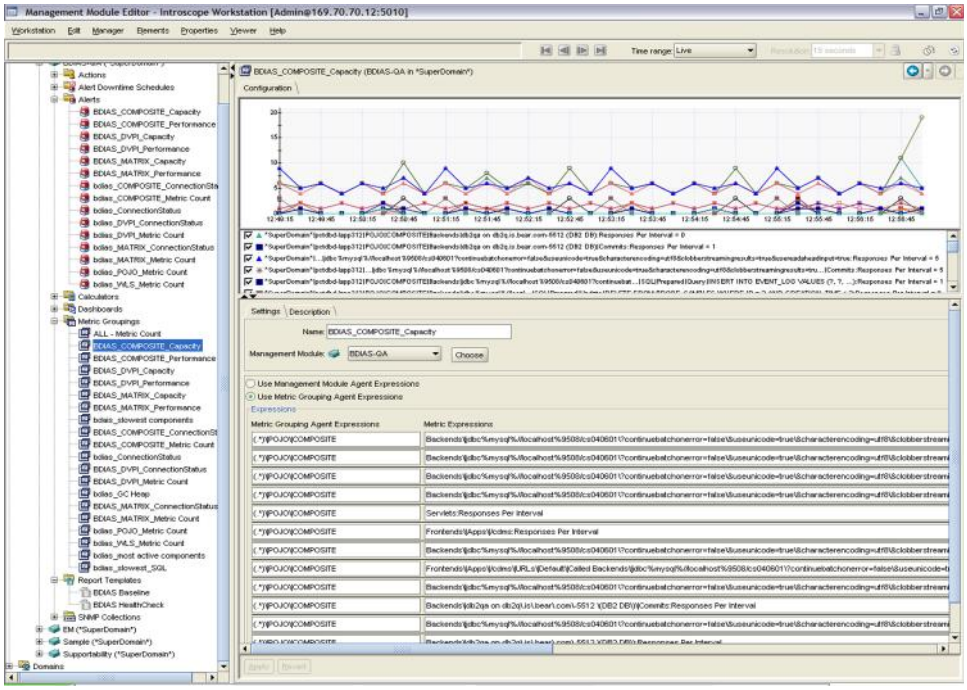
Now drag the specific metric of interest to the Management Module Editor tab at the bottom of the screen and wait for the application to pop to the foreground. The following screenshot illustrated the situation just before the screen pops:



Drag onto the Management Module Editor, wait for that window to 'pop' --

FIGURE 14: BUILDING A METRIC GROUP – DRAG AND POP

With the Management Module editor in view, drag to the open row of the metric grouping and release. The following screenshot is the result after the metric has been successfully dragged-n dropped:

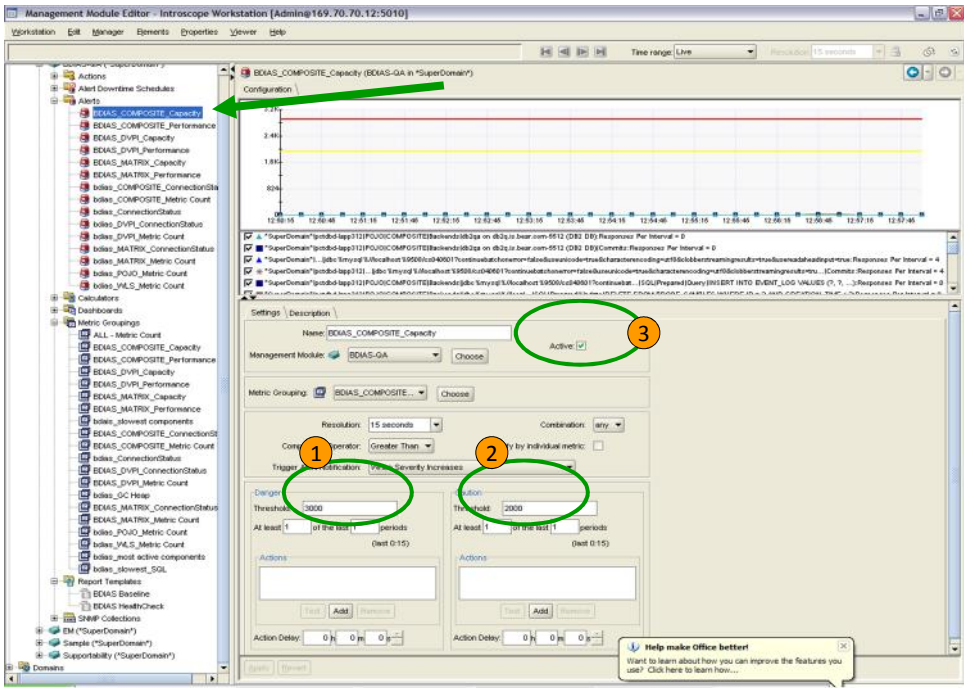


Drop the metric into the previously opened metric grouping

FIGURE 15: BUILDING A METRIC GROUP – LIST OF METRICS

Now you can use <ALT><TAB> to pop the Investigator window, hit the back button (upper right) and return to the Search tab. Continue to locate, drag-n-drop until your metric grouping is complete.

Once the metric group is established, right-click the metric group and select "Simple Alert" and then fill in the appropriate values for the threshold, and make sure to active it. The following screenshot summarizes the situation after the change has been accepted.



Right-click the metric grouping, select “Simple Alert” fill in the thresholds, activate.

FIGURE 16: CONFIGURING AN ALERT THRESHOLD

You will repeat this process for as many metric groups and alert thresholds as you might need. Bear in mind that most of the alerts being defined are to help the dashboard effort, and will not be delivered into the Trouble Management system, as an alert to be handled by Operations. Which alerts may be needed is outside the scope of the audit. Or goal now is to simply identify the most reliable metric groups (and underlying metrics) and which to define an alert threshold.

ALERT HEIRARCHY

Most projects begin with the definition of alerts, even as this is not the best way to start. There is no shortcut to finding the correct alerts and existing alerts are usually excessive. This is always the case when APM is replacing an earlier technology. And usually the case when APM is a ‘Greenfield’ project. Alerts are just what folks associate with ‘monitoring’. You need to capture this ambition and harness it to do APM==management. As we have done so far, we want to emphasize understanding the metrics, identify the more significant ones and then follow this with the alert definition.

You could make a formal assessment of the alerting but this is a trap as it assumes that all the alerts are valid. You should instead do an incident analysis (Chapter 3 in APM Best Practices) to help identify the types of problems you want to alert on, and then make sure that you have visibility and thresholds to capture those situations. This approach lets you focus on the alerts that are needed – not simply the alerts that are available. When all you have is a hammer, everything looks like a nail. Make sure you understand all the ‘fasteners’ that are in use but focus on the ones that are needed, as indicated by the incident list.

Of course, if you are doing an audit, it’s a little too late to step back in time a few weeks to justify what alerts are needed. Instead, we want to create a small list of alerts that you will immediately need, and that will also serve as a repeatable exercise to help folks understand how to configure them. This is simply Availability, Performance and Capacity, or APC. And if time or testing is short and folks otherwise cannot decide which alerts are necessary or not, you will at least have a set of alerts that are validated and completely correct. You can always add more but don’t leave before the basics (APC) are established.

The following Alert Hierarchy should be implemented as part of the Application Audit: [needs more]

- AVAILABILITY (summary alert)
  - CONNECTIVITY_ALERT (Connectivity metric)
  - AVAILABILITY_ALERT (Summary Alert)
    - APP_AVAILABILITY_ALERT (Metric Count)
    - Other availability metrics as appropriate

- PERFORMANCE (summary alert)
  - PERFORMANCE_ALERT (Key Servlet Response Times)
  - PERFORMANCE_ALERT (key JMS Response Times)
  - PERFORMANCE_ALERT (Key SQL Response Times)

- CAPACITY (summary alert)
  - CAPACITY_ALERT (Key Servlet Invocations)
  - CAPACITY_ALERT (Key JMS Invocations)
  - CAPACITY_ALERT (Key SQL Invocations)

TABLE 17: ALERT NAMING CONVENTION

Now sometimes you will not have enough time or sufficient visibility to define and validate alerts for Performance and Capacity. There may be problems with load testing, or not enough time in the schedule for one last load test. So focus on setting up a few metric groups for availability, threshold them and then assemble them into a Summary Alert for Availability – and you have established the pattern that they will follow for the rest of their APM experience.

The following graphic summarizes how the Management Module looks after the best practice has been followed. Three Summary Alerts, or placeholders, ready to forward alerts or to establish a basic dashboard.

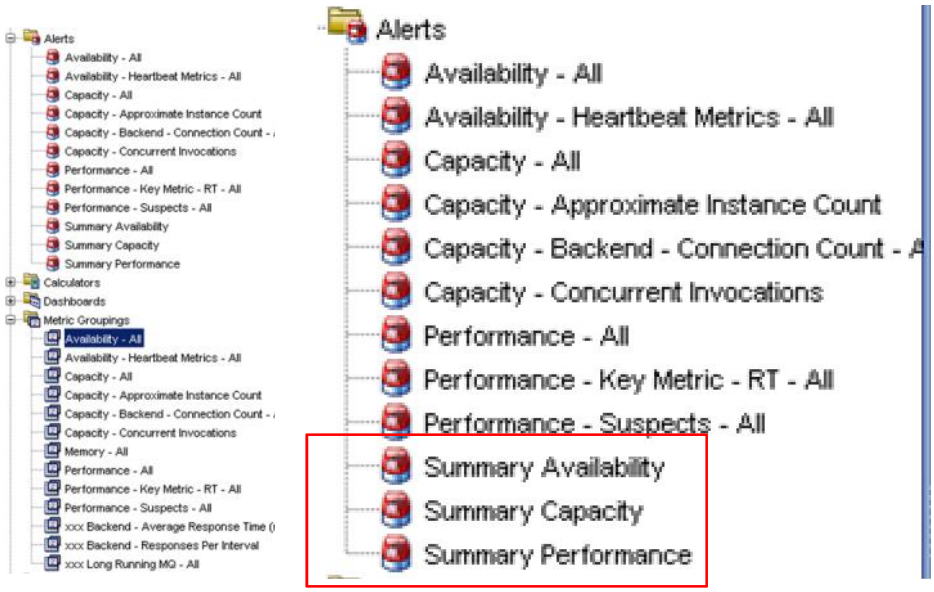


FIGURE 18: EXAMPLE ALERT HIERARCHY FOR THRESHOLDED METRICS AND SUMMARY ALERTS

Preparing Dashboards

For the dashboard, simple is best. You want to create a big placeholder for the various applications, even beyond what you have examined as part of the audit. "Nature abhors a vacuum." – Aristotle

Likewise, folks will look at a dashboard full of grayed-out placeholders… and work to 'light them up'. This will highlight the benefits of consistent naming and to start the hierarchal dashboards considerations that will make it easy for folks to navigate when the alerting really gets established.

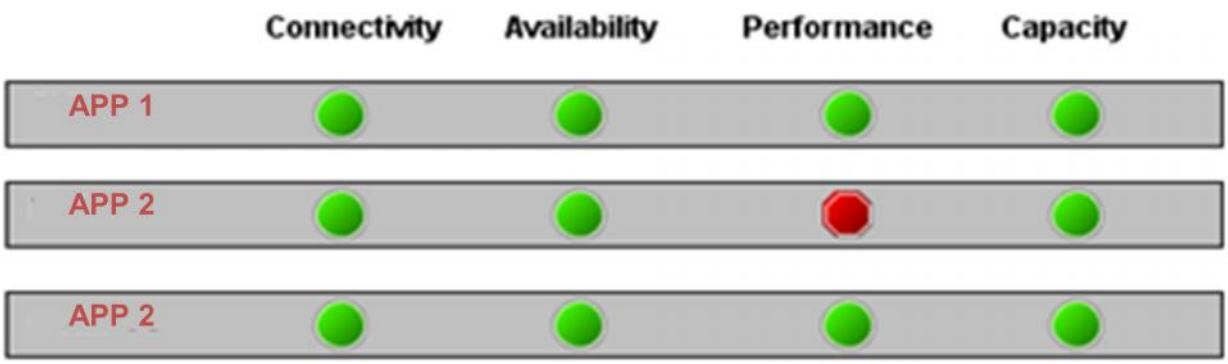


FIGURE 19: EXAMPLE DASHBOARD OF THRESHOLDED SUMMARY METRICS (ALERTS)

You should make a quick template of a dashboard with the client logo, link those in to the "APP 1", etc. name. Leave those completely blank. This is, after all, only an audit.

Preparing a Baseline Report

The metric groups that define Availability, Performance and Capacity can be repurposed for these baseline reports but these may not be sufficient in the number of metrics, to be useful for supporting version analysis and other trending activities. In fact, a broader baseline will be the starting point from which you will later select the metrics that are the best candidates for thresholds. Some simply prepare a metric grouping for "Component Response" and "Invocations" and then drag-n-drop the metrics you find during search and ordering into the appropriate metric grouping. Since we will be using a table, it doesn't matter if the *dynamic range* of the metrics, are not compatible. Of course, this type of metric group may be useless for a graph. If you were to have a metric group containing a response time metric (3 seconds) and a GC metric (450,000 KBytes), you will never see the response time metric on the graph.

In the example that follows, we have identified a significant metric for each of the component types that a specific application presents. While these are response time values, you have actually selected the component from the highest invocation (responses per interval) value search, and now are tracking the response time of that *most active* component.

The motivation for this should be obvious. The most active component will also be the most sensitive to a change in response time. So when we improve (decrease) response time, the overall response time of the application should improve. Conversely, if we do something that degrades the response time (increase) of the component, we will likely severely impact the overall performance of the app.

Two exceptions to this relationship are 1) when the change has no apparent effect, and 2) a bottleneck exists which is masking or absorbing the impact of the change. This is why the other search, on highest response times, is also necessary. In this we are effectively scouting the application looking for bottlenecks. This is not reflected directly in the magnitude of the response time value but in how much it varies, in relationship to load. This analysis also requires that we examine candidate components in the context of the transaction times we are looking to improve, which comes from the *Application Baseline*.

Thus we can see we really need all three baselines, configuration, application and performance, to help us correctly and accurately characterize application performance. The *configuration baseline* confirms the compatibility of the monitoring configuration and helps

identify tuning opportunities for the JVM and app server configuration settings. The *application baseline* surveys the transactions and reveals component relationships and overall design patterns. It also confirms that we have adequate visibility into the transaction. The *performance baseline* identifies the key contributors to performance (response time) and capacity (invocations).

Baseline - Component Response

Component Response Time (overview)

Resource:Metric	Mean	Min	Max	MMin	MMax	Count
EJB|Message-driven:Average Method Invocation Time (ms)	1,284	9	106,942	0	607,856	12,775
EJB|Session:Average Method Invocation Time (ms)	1,851	1	279,491	0	1,369,006	9,693
JDBC:Average Query Time (ms)	950	0	37,218	0	83,313	28,214
JDBC:Average Update Time (ms)	79	0	13,093	0	200,896	4,221
JMS|Message Consumer:Average Method Invocation Time (ms)	57,109	567	143,981	2	716,873	66,412
JMS|Message Listener:Average Method Invocation Time (ms)	11,828	0	168,678	0	1,181,781	5,476
JMS|Queue Sender:Average Method Invocation Time (ms)	513	0	21,018	0	278,362	3,604
JMS|Topic Publisher:Average Method Invocation Time (ms)	36	1	32	0	179	36
JTA:Average Method Invocation Time (ms)	203	0	29,949	0	485,008	14,979
Servlets:Average Response Time (ms)	8,884	352	13,232	0	54,935	13

Start: 5/9/06 9:00 AM End: 5/17/06 2:00 PM

Metric Grouping: Component Response Time (ms)
Agents: ftbhvt01|WebLogic|wfttf_srv_01

FIGURE 20: EXAMPLE BASELINE REPORT

While we are showing a single baseline report it is not unusual to have additional baseline reports focused on specific components or monitoring configurations. You will always need the example baseline report shown here in order to track the overall performance characteristics, especially when comparing releases or configuration changes. If you have a portal configuration, or resources like MQ or DataPower or Mainframe – these will also benefit from a more tailored baseline report, especially when these are being managed by a separate group. In fact, for any type of resource or monitoring tool, being able to produce a baseline of 'normal' performance is essential to bringing that tool and information into the APM practice.

The meanings of the value values that are represented in the table are depicted in the following figure:

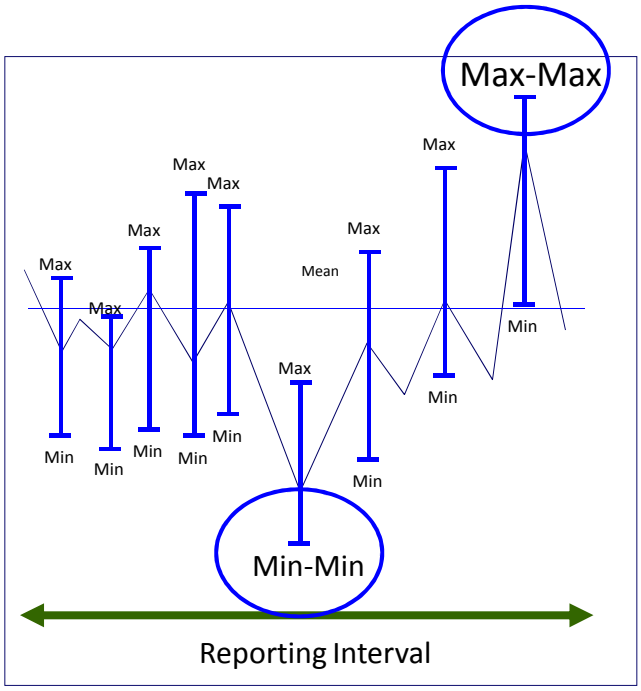


FIGURE 21: RELATIONSHIP OF AVERAGE, MAXIMUM AND MAX-MAX

Each data point is the result of averaging the real-time activity of a particular metric. However, when the metric value is updated, currently every 15 seconds, we are also sending the maximum and minimum values that were encountered during that 15 second interval. So each data point, representing the average value, also has a corresponding maximum and minimum value. You can see this in the Introscope display by turning on "Min-Max". For the reporting interval we also collect the maximum and minimum values of that report interval and present that as the Max-Max and Min-Min. These are used to assess the quality of a metric when determining which metrics to use for a threshold and alert, as was discussed in the earlier section Delivery – Activities – Determining Thresholds.

While this approach reveals the variability what a particular metric is experiencing, it is not the same as a Standard Deviation or Variance function. These are the more traditional way to look at a population but we simply do not have the time to calculate these quantities. To maintain real-time performance of the APM platform we are limited to integer math.

Statistics alone do not tell us when a Standard Deviation is acceptable or unacceptable – only that the variance can be measured. The Max-Max and Min-Min are perhaps less elegant and the correlation guidelines (see section Delivery – Activities – Determining Thresholds - Subjective Conclusion Matrix for Audit) more arbitrary – however the result seems to be the same, in terms of identifying the most consistent metric for threshold.

Wrap-up Meeting

Resist all attempts to deliver full recommendations that same day you are delivering the Application Audit. Take the time to review your results and recommendations. If the remaining time or scheduling is too difficult, setup a follow-up call and use LiveMeeting to review your presentation. You want to make this exercise something that is useful and predictable, in terms of time commitments by the customer. You will build a much better relationship if you keep scope and show up periodically. Don't try to do everything in a single visit.

I've included a variety of reporting outlines for you to follow, when you get to that section. So long as you have got plenty of screen shots – it will be pretty easy. In general, you don't want to say anything that you cannot also show them why and how you know that to be relevant. Do not rely on a live exercise of the workstation to show results, as part of your wrap-up meeting. This is not a demo of the software! This is an audit of the application and it is very likely going to be repeated in the future. You must ensure than you have all the artifacts preserved so that the next practitioner can track the progress of any configuration changes, performance tunings or new functionality.

You should allow 30 minutes for a 'good' report and 1 hour if there are problems. Typically, there is always something to be adjusted. The main questions you want to address are as follows:

- Is the application stable? What are the major suspects?
- Will the current application scale? How much more before performance degrades?

- Can the application be managed operationally, via the dashboards and reports prepared?
- Is the monitoring configuration optimal or do visibility gaps remain?

Written Report

For longer engagements you will likely need to prepare a written report. This is where your discipline is most important – do not say ANYTHING that you cannot support with a screenshot or other summary data. This will come back to bite you, if your recommendations are not successful. In general, you want to report on what you have observed, what the good software engineering recommendations are or what generally-known performance criteria demand. If you make a configuration recommendation, include all of your artifacts.

Please also consider having a more experienced practitioner review your document if you have any concerns.

What to Look For

As you conduct the Application Audit, you are going to 'see things'. What do they mean? This section will give you some starting points to consider. The ultimate arbitrator will be the experience of you and your peers. Every application is different. By using the Application Audit process we can ensure consistency of the monitoring configuration validation and resulting performance analysis. Over time, we can look forward to this consistency as the foundation for meaningful discussions about what good and bad performance characteristics really are.

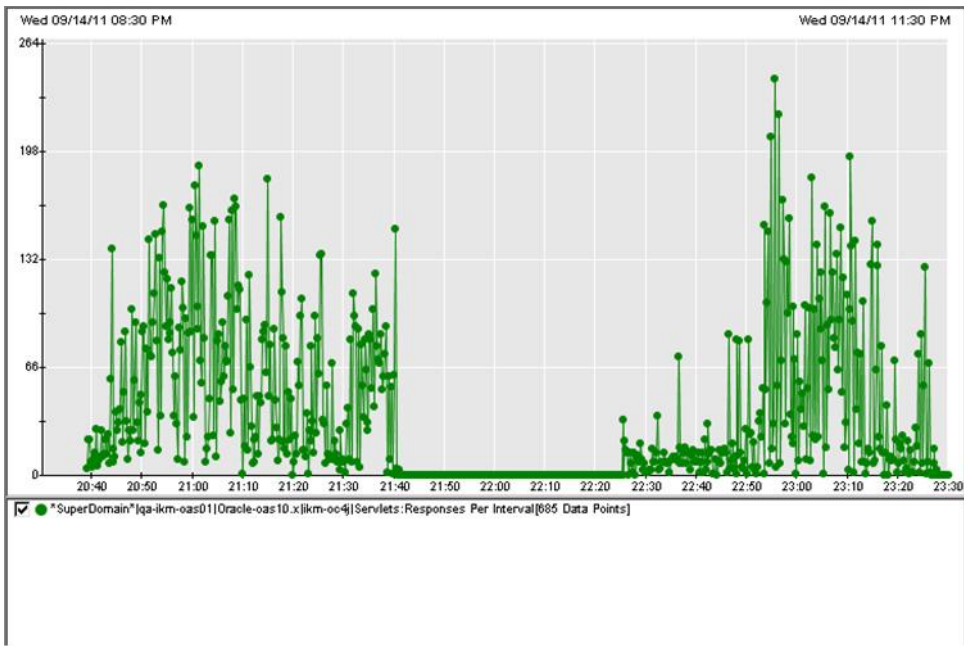
Key Performance Metrics and Behaviors

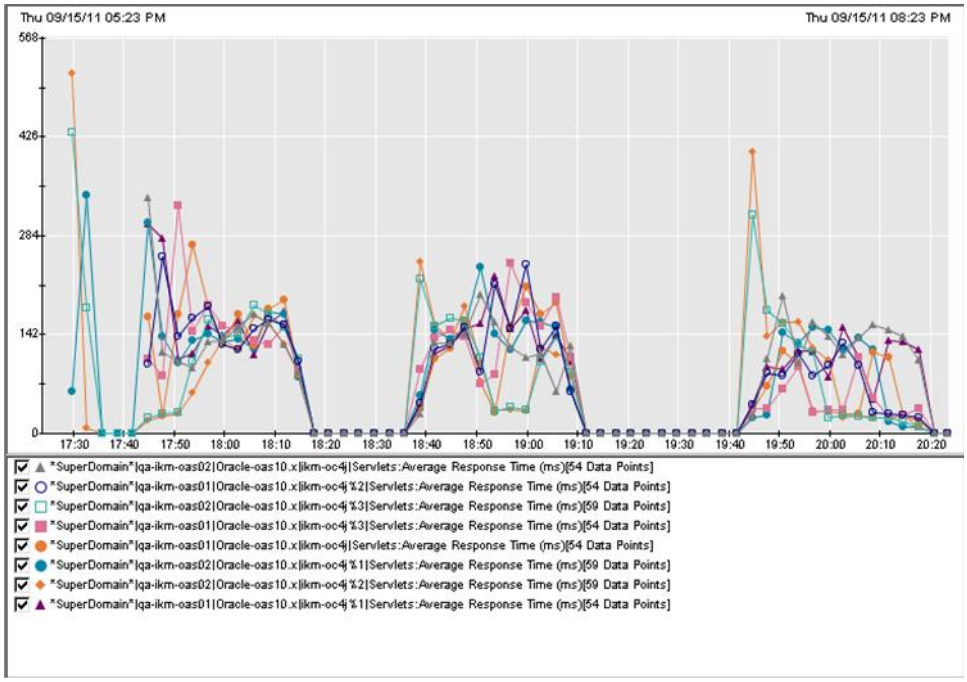
This is a gallery of various situations, coming from live audits (where possible). Pass and Fail, with explanations as appropriate. This section is expected to grow when the pace of audits increases.

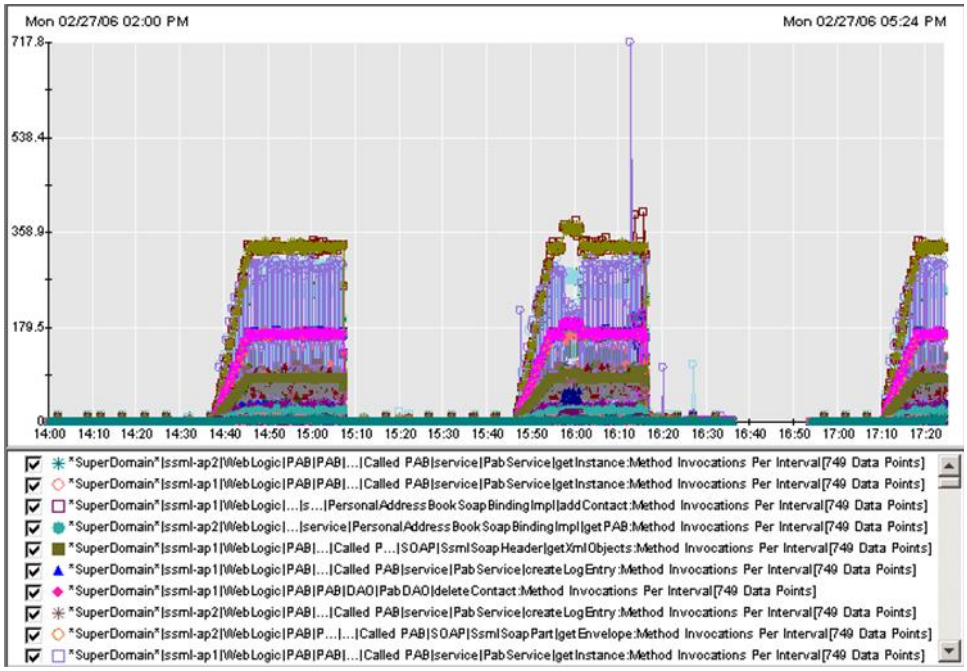
LOAD REPRODUCIBILITY

Pass

Fail

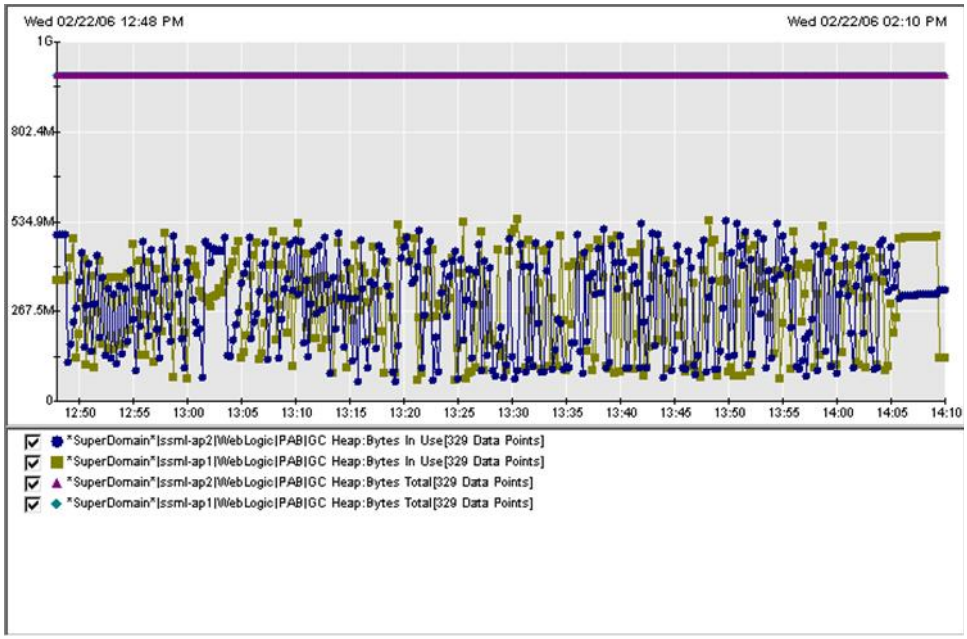


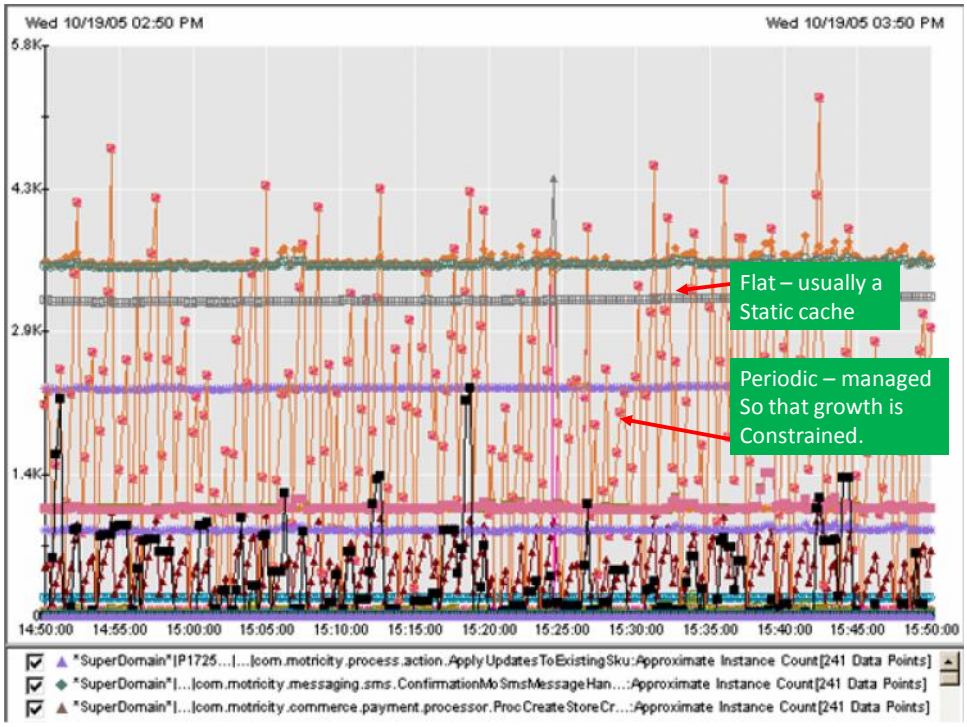

Thu 09/15/11 05:23 PM
Thu 09/15/11 08:23 PM
568
426
284
142
0
17:30 17:40 17:50 18:00 18:10 18:20 18:30 18:40 18:50 19:00 19:10 19:20 19:30 19:40 19:50 20:00 20:10 20:20
SuperDomain|qa-ikm-oas02|Oracle-oas10.x|ikm-oc4j|Servlets:Average Response Time (ms)[54 Data Points]
SuperDomain|qa-ikm-oas01|Oracle-oas10.x|ikm-oc4j%2|Servlets:Average Response Time (ms)[54 Data Points]
SuperDomain|qa-ikm-oas02|Oracle-oas10.x|ikm-oc4j%3|Servlets:Average Response Time (ms)[59 Data Points]
SuperDomain|qa-ikm-oas01|Oracle-oas10.x|ikm-oc4j%3|Servlets:Average Response Time (ms)[54 Data Points]
SuperDomain|qa-ikm-oas01|Oracle-oas10.x|ikm-oc4j|Servlets:Average Response Time (ms)[54 Data Points]
SuperDomain|qa-ikm-oas02|Oracle-oas10.x|ikm-oc4j%1|Servlets:Average Response Time (ms)[59 Data Points]
SuperDomain|qa-ikm-oas02|Oracle-oas10.x|ikm-oc4j%2|Servlets:Average Response Time (ms)[59 Data Points]
SuperDomain|qa-ikm-oas01|Oracle-oas10.x|ikm-oc4j%1|Servlets:Average Response Time (ms)[54 Data Points]


Mon 02/27/06 02:00 PM
Mon 02/27/06 05:24 PM
717.8
538.4
358.9
179.5
0
14:00 14:10 14:20 14:30 14:40 14:50 15:00 15:10 15:20 15:30 15:40 15:50 16:00 16:10 16:20 16:30 16:40 16:50 17:00 17:10 17:20
SuperDomain|ssml-ap2|WebLogic|PAB|PAB|...|Called PAB|service|PabService|getInstance:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|PAB|PAB|...|Called PAB|service|PabService|getInstance:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|...|s...|PersonalAddressBookSoapBindingImpl|addContact:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap2|WebLogic|...|service|PersonalAddressBookSoapBindingImpl|getPAB:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|PAB|...|Called P...|SOAP|SsmlSoapHeader|getXmlObjects:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|PAB|...|Called PAB|service|PabService|createLogEntry:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|PAB|PAB|DAO|PabDAO|deleteContact:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap2|WebLogic|PAB|...|Called PAB|service|PabService|createLogEntry:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap2|WebLogic|PAB|P...|...|Called PAB|SOAP|SsmlSoapPart|getEnvelope:Method Invocations Per Interval[749 Data Points]
SuperDomain|ssml-ap1|WebLogic|PAB|PAB|...|Called PAB|service|PabService|getInstance:Method Invocations Per Interval[749 Data Points]

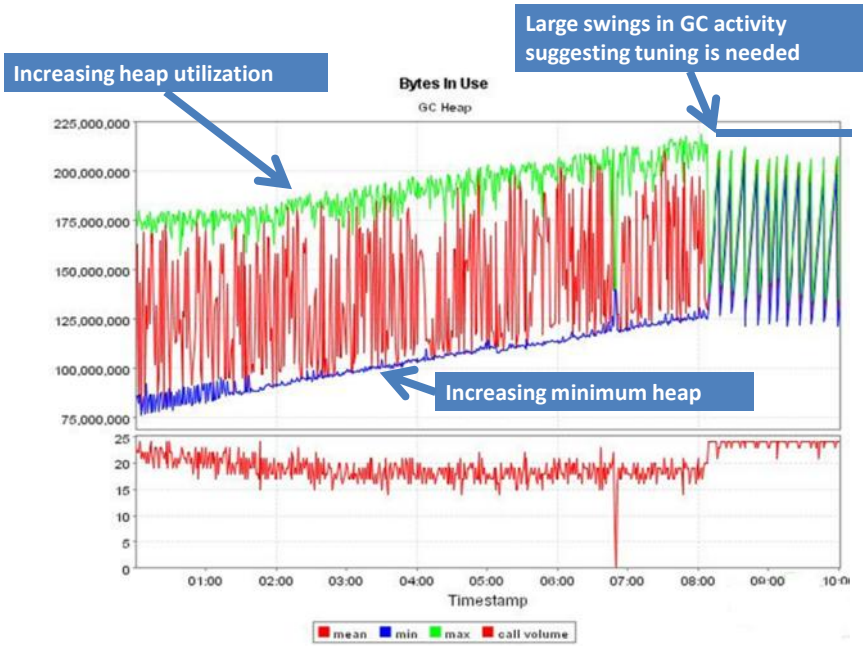
MEMORY MANAGEMENT – GC/HEAP, INSTANCE COUNTS

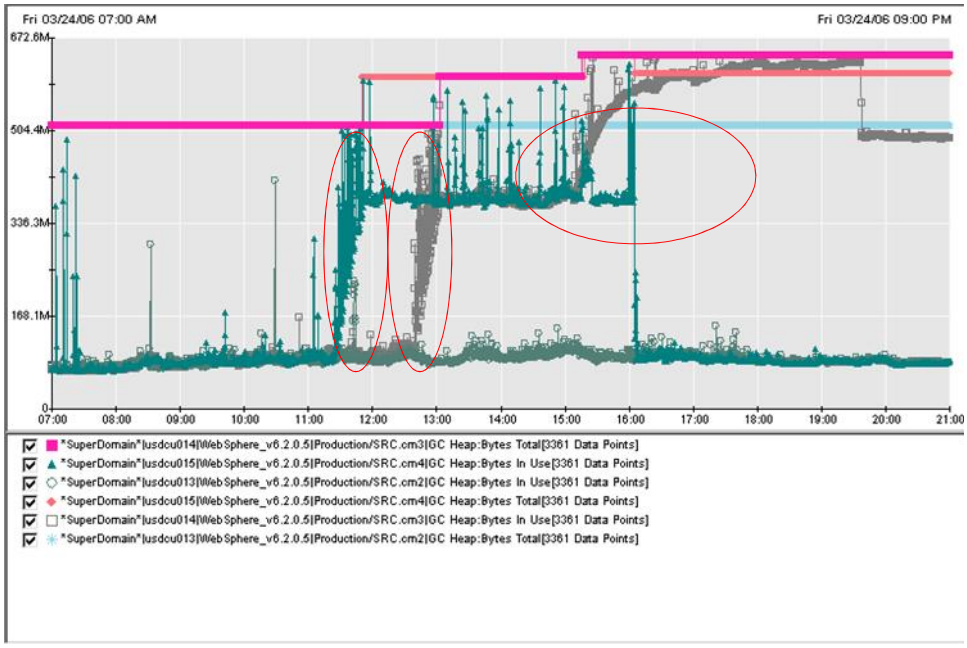
Pass



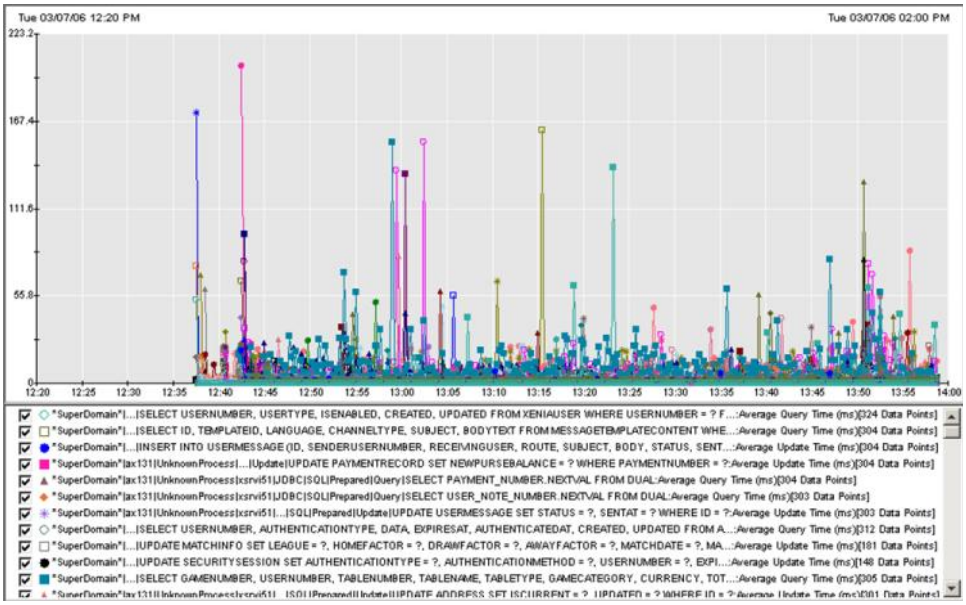


Fail


Increasing heap utilization
Large swings in GC activity suggesting tuning is needed
Increasing minimum heap
Bytes In Use
GC Heap
225,000,000
200,000,000
175,000,000
150,000,000
125,000,000
100,000,000
75,000,000
25
20
15
10
5
0
01:00
02:00
03:00
04:00
05:00
06:00
07:00
08:00
09:00
Timestamp
mean
min
max
call volume

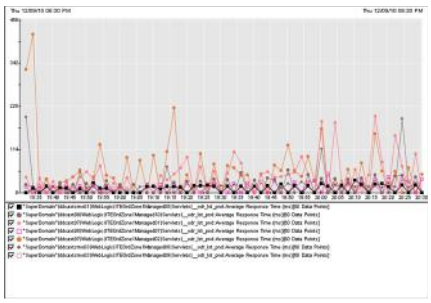

Fri 03/24/06 07:00 AM
Fri 03/24/06 09:00 PM
672.6M
504.4M
336.3M
168.1M
0
07:00
08:00
09:00
10:00
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
SuperDomain|usdcu014|WebSphere_v6.2.0.5|Production/SRC.cm3|GC Heap:Bytes Total[3361 Data Points]
SuperDomain|usdcu015|WebSphere_v6.2.0.5|Production/SRC.cm4|GC Heap:Bytes In Use[3361 Data Points]
SuperDomain|usdcu013|WebSphere_v6.2.0.5|Production/SRC.cm2|GC Heap:Bytes In Use[3361 Data Points]
SuperDomain|usdcu015|WebSphere_v6.2.0.5|Production/SRC.cm4|GC Heap:Bytes Total[3361 Data Points]
SuperDomain|usdcu014|WebSphere_v6.2.0.5|Production/SRC.cm3|GC Heap:Bytes In Use[3361 Data Points]
SuperDomain|usdcu013|WebSphere_v6.2.0.5|Production/SRC.cm2|GC Heap:Bytes Total[3361 Data Points]

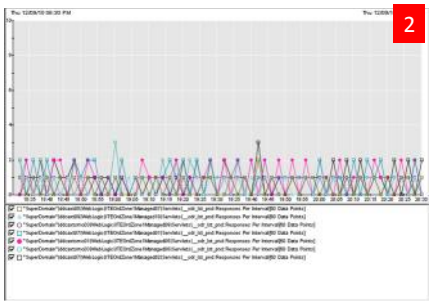
Pass



Good Servlet (lightly used)

Servlet - odr_lst_pnd

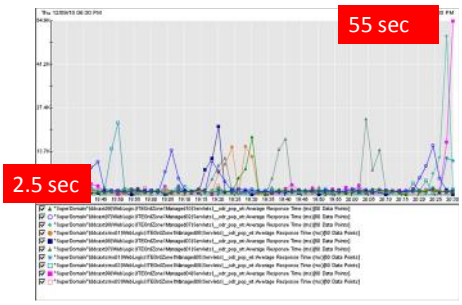


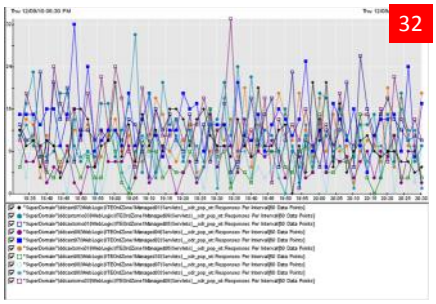


Fail

Bad Servlet (moderately used)

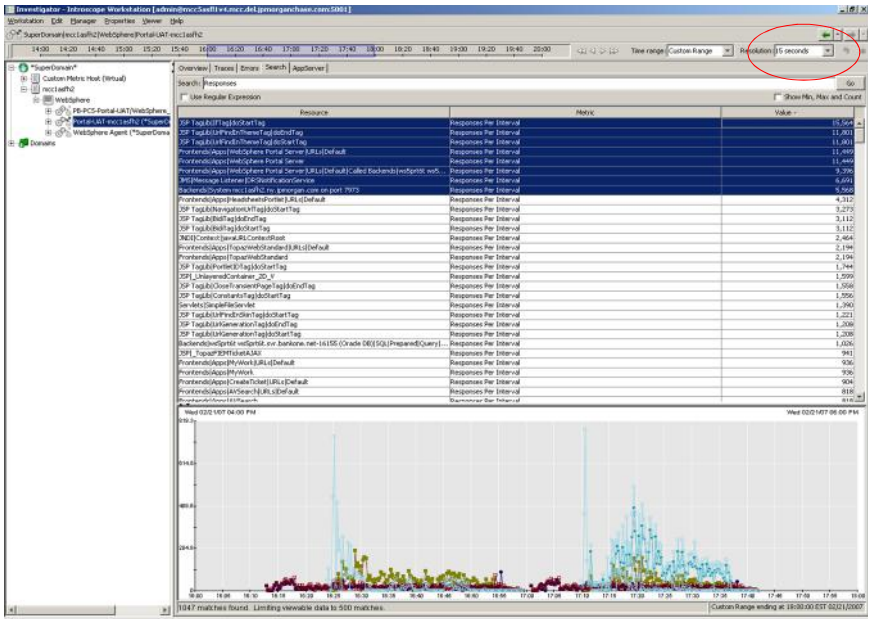
Servlet - ord_pop_wt





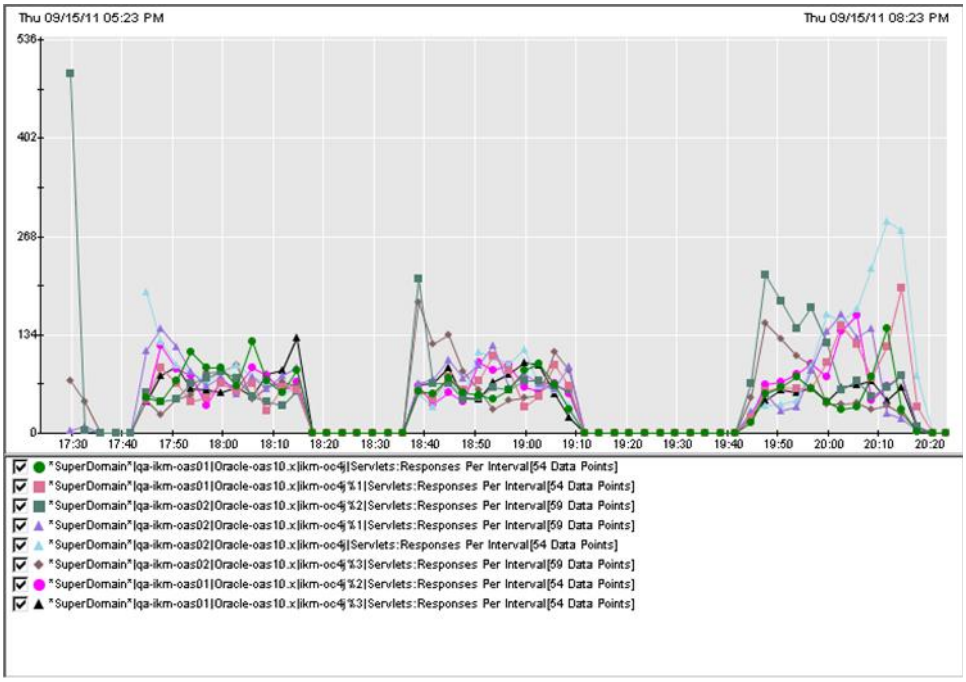
INVOCATION RATES

Pass



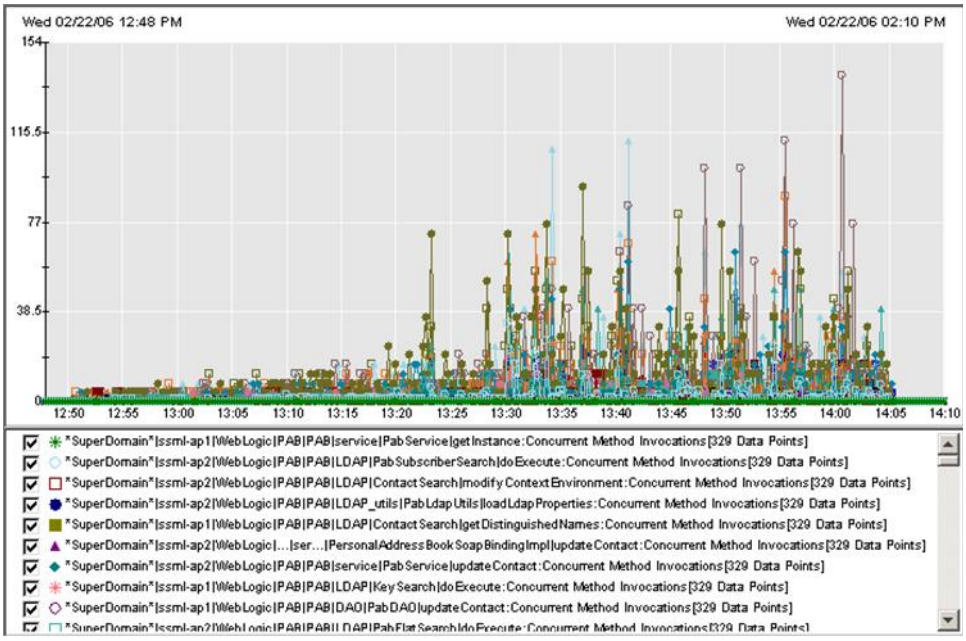
Any metrics exceeding 5000 invocations per 15 seconds?

Fail

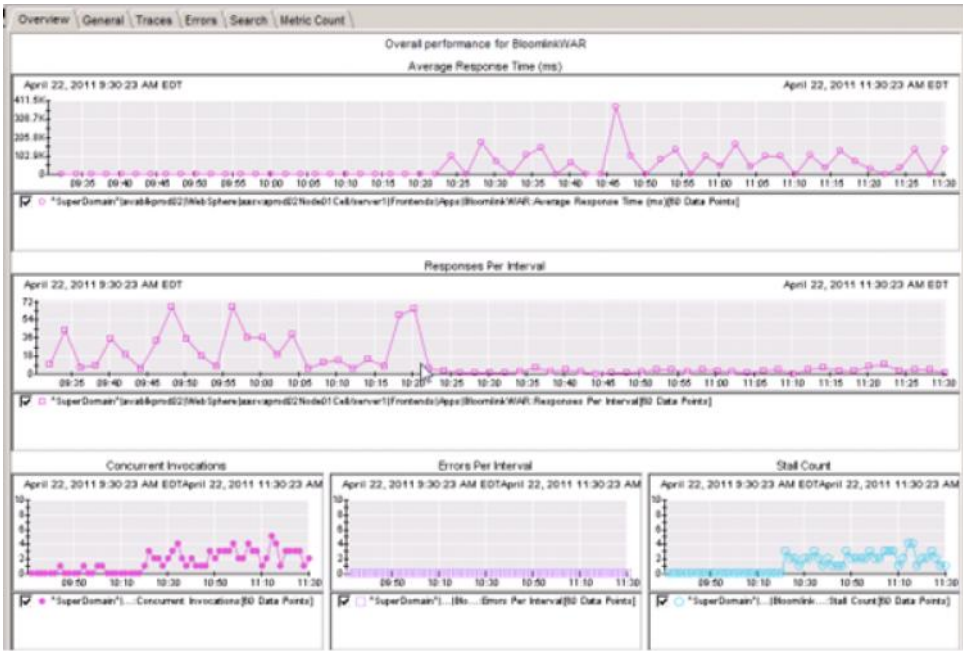


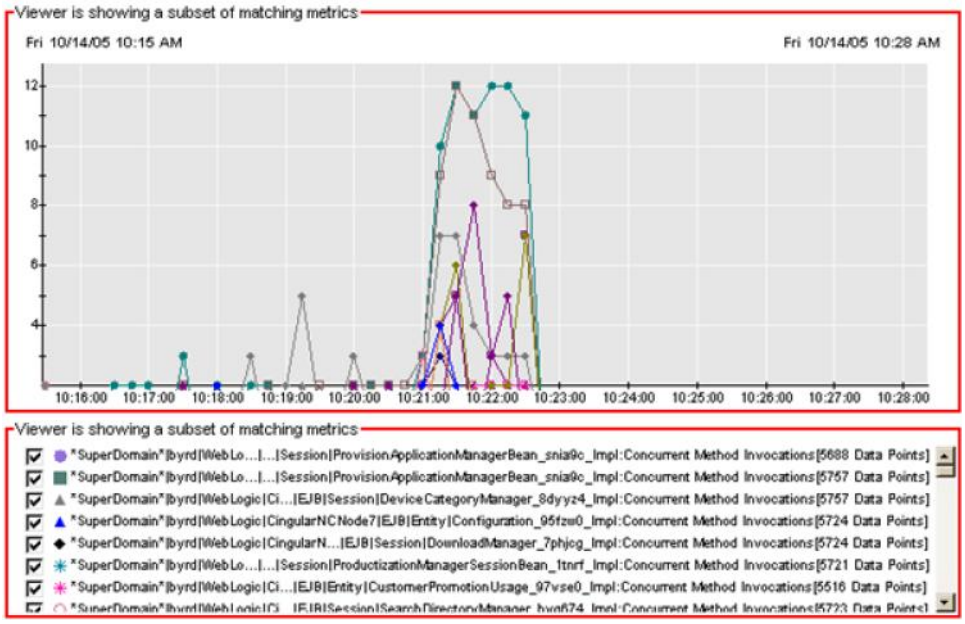
Pass

The concurrent response is appropriate but the application is already saturated (cannot scale). This occurs at 13:20pm. The response time will degrade >300% by 13:30.



Fail



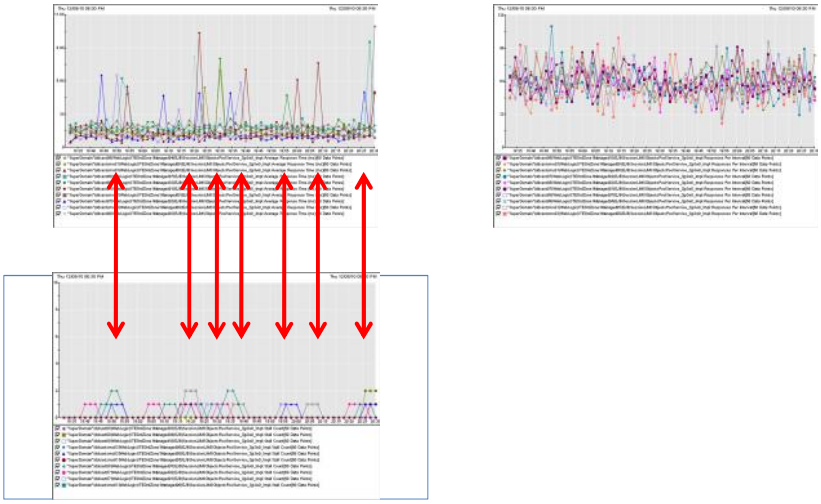

Viewer is showing a subset of matching metrics
Fri 10/14/05 10:15 AM
Fri 10/14/05 10:28 AM
12
10
8
6
4
10:16:00
10:17:00
10:18:00
10:19:00
10:20:00
10:21:00
10:22:00
10:23:00
10:24:00
10:25:00
10:26:00
10:27:00
10:28:00
Viewer is showing a subset of matching metrics
SuperDomain|byrd|WebLo...|...|Session|ProvisionApplicationManagerBean_snia9c_Impl:Concurrent Method Invocations[5688 Data Points]
SuperDomain|byrd|WebLo...|...|Session|ProvisionApplicationManagerBean_snia9c_Impl:Concurrent Method Invocations[5757 Data Points]
SuperDomain|byrd|WebLogic|Ci...|EJB|Session|DeviceCategoryManager_8dyyz4_Impl:Concurrent Method Invocations[5757 Data Points]
SuperDomain|byrd|WebLogic|CingularNCNode7|EJB|Entity|Configuration_95fzw0_Impl:Concurrent Method Invocations[5724 Data Points]
SuperDomain|byrd|WebLogic|CingularN...|EJB|Session|DownloadManager_7phjcg_Impl:Concurrent Method Invocations[5724 Data Points]
SuperDomain|byrd|WebLo...|...|Session|ProductizationManagerSessionBean_1tnrf_Impl:Concurrent Method Invocations[5721 Data Points]
SuperDomain|byrd|WebLogic|Ci...|EJB|Entity|CustomerPromotionUsage_97vse0_Impl:Concurrent Method Invocations[5516 Data Points]

STALLS

Pass

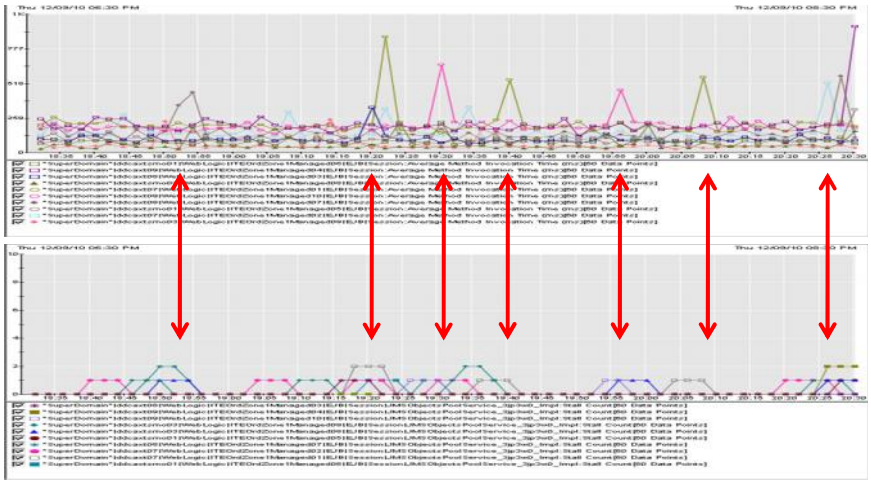
Stalls recover but application suffers.

JMS Object Pool Service

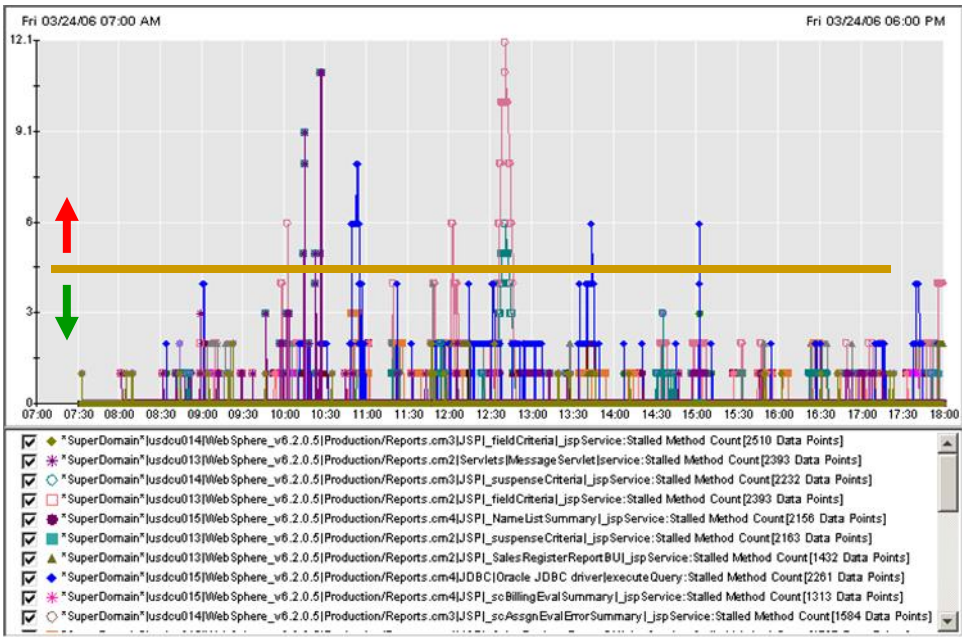


Stalls recover but EJB response time suffers

EJB-Session – JMS Stalls

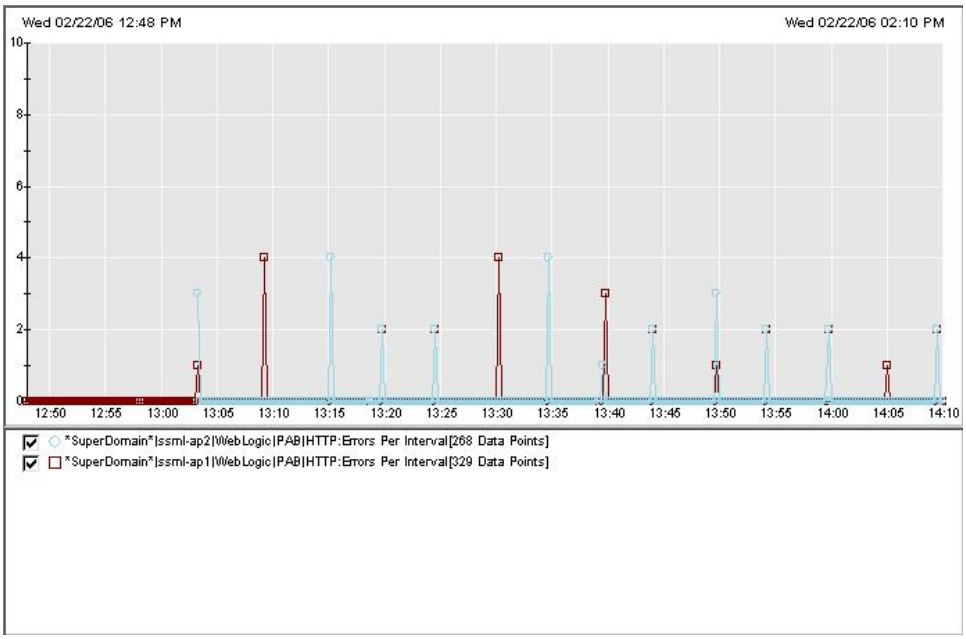


Fail


Fri 03/24/06 07:00 AM
Fri 03/24/06 06:00 PM
12.1
9.1
6
3
0
07:00 07:30 08:00 08:30 09:00 09:30 10:00 10:30 11:00 11:30 12:00 12:30 13:00 13:30 14:00 14:30 15:00 15:30 16:00 16:30 17:00 17:30 18:00
"SuperDomain"|usdcu014|WebSphere_v6.2.0.5|Production/Reports.cm3|JSPI_fieldCriteria|_jspService:Stalled Method Count[2510 Data Points]
"SuperDomain"|usdcu013|WebSphere_v6.2.0.5|Production/Reports.cm2|Servlets|MessageServlet|service:Stalled Method Count[2393 Data Points]
"SuperDomain"|usdcu014|WebSphere_v6.2.0.5|Production/Reports.cm3|JSPI_suspenseCriteria|_jspService:Stalled Method Count[2232 Data Points]
"SuperDomain"|usdcu013|WebSphere_v6.2.0.5|Production/Reports.cm2|JSPI_fieldCriteria|_jspService:Stalled Method Count[2393 Data Points]
"SuperDomain"|usdcu015|WebSphere_v6.2.0.5|Production/Reports.cm4|JSPI_NameListSummary|_jspService:Stalled Method Count[2156 Data Points]
"SuperDomain"|usdcu013|WebSphere_v6.2.0.5|Production/Reports.cm2|JSPI_suspenseCriteria|_jspService:Stalled Method Count[2163 Data Points]
"SuperDomain"|usdcu013|WebSphere_v6.2.0.5|Production/Reports.cm2|JSPI_SalesRegisterReportBU|_jspService:Stalled Method Count[1432 Data Points]
"SuperDomain"|usdcu015|WebSphere_v6.2.0.5|Production/Reports.cm4|JDBC|Oracle JDBC driver|executeQuery:Stalled Method Count[2261 Data Points]
"SuperDomain"|usdcu015|WebSphere_v6.2.0.5|Production/Reports.cm4|JSPI_scBillingEvalSummary|_jspService:Stalled Method Count[1313 Data Points]
"SuperDomain"|usdcu014|WebSphere_v6.2.0.5|Production/Reports.cm3|JSPI_scAssgnEvalErrorSummary|_jspService:Stalled Method Count[1584 Data Points]

ERRORS

Pass



Fail

Components invocations go to zero as errors occur

REPORTING

Screenshots

Screenshots are always more efficient then writing down enough details to actually reproduce the view you are currently interested in. It also makes it completely simple to annotate and later communicate and mentor the APM team. It also makes it easy to get help, via the WIU, via email. Everybody loves a puzzle and a picture says a lot – compared to your pressured written description of the problem. The APM workstation today also presents a couple of features to copy just the graphs and this will make for a neater presentation or report.

Introscope Reports

For deeper analysis and comparison of events, the Introscope report is preferred. Here you can collect numbers in a tabular form and then use spreadsheets or scripting to make the analysis. This is absolutely necessary when do the EM Load Testing (Solution Certification) but here we are actually planning to do testing and will take the time to set up the appropriate metrics groups and automate the report generation. For most clients, the significant gap is any structure around testing and the use of reporting in general. They are so sold on the use of dashboards that they set-aside the fundamentals of basic, reliable reporting.

The real gap is that they do not know how to decide what to report on. Now, with the *APM Best Practices*, there isn't an excuse anymore. They follow the baseline process and they end up with a manageable set of metrics and some basic understanding of the application performance characteristics. They will know what to trend and compare.

Wrap-up Presentation

The *Application Audit* is a core Best Practice as it exercises most of the skills and processes needed to confirm a monitoring configuration, identify the key performance metrics, and prepare reports and dashboards to make the production experience with APM most effective. It is something that every client should be capable of doing on their own and often this exercise is the first real appreciation of how to use APM technology in a deliberate and meaningful fashion. You need to relate the relative ease in getting this accomplished, the time needed by various stakeholders and richness of the findings that APM visibility will reveal.

Always remember NOT to make any broad pronouncements about what may be broken or what should be fixed. You want to emphasize what the visibility reveals, what is consistent with other types of applications, and what is not. The final decisions need to be taken with a full understanding of the application architecture and design – and that is often outside the scope of a few days of exercising the application.

In the following outline, the term "slide" is interchangeable with "section". Try to fit the points on a single slide but use as many slides as are appropriate. The fewer - the better. If you have additional examples that you want to highlight, then move these to an appendix. Try to keep the main presentation as short as practical. You will want to cover the following topics during your wrap-up:

Slide 1 – What was done

- Performance Audit of Application XXX
- Which load scenarios or use cases that were explored
- The number of metrics per JVM experienced
- Total testing time and analysis time
- Any impediments

Slide 2 – Configuration Baseline details

- What agent configurations were explored
- How many metrics did each produce
- Any incompatibilities experienced
- Results of overhead testing (if attempted)

Slide 3 – Application Baseline details

- What transaction types were exercised
- Was instrumentation sufficient to support triage?
- Any unusual transaction traces collected?

- (Follow with example screen shots, as appropriate)
- (Be sure to include a screenshot of a 'good' level of instrumentation, "too much' and 'too little')

Slide 4 - Performance Baseline Details

- What are the metrics that best represent Availability, Performance and Capacity (APC)?
- What thresholds and alerts were defined (list significant examples only)
- What metric groupings were established (list significant examples only, summary alerts, etc.)
- What baseline reports were configured
- What hierarchical dashboards were configured (include screenshots)

Slide 5 – Results of Stress-to-Failure Load Test (usually 2 hrs or more duration)

- Screenshot of response time degradation
- Screenshot of Invocations degradation
- Calculation of target capacity
- Screenshot of dashboard validating that key thresholds fired when capacity exceeded

Slide 6: Concerns

- What did you observe that appeared to deviate from typical applications? Just try and get all of your concerns on one slide.

Slide 7 : Recommendations

- A list of what you recommend they should do to address the concerns and whom they should collaborate with.

Slide 8: Additional Details

- A summary of what you observed, that led to each of your concerns. And also what you saw that appeared 'typical'. Screenshots should be used here with annotations (circles and arrows) specifically highlighting the issue.

Written Report and Recommendations

Within reason, please do not recommend ANYTHING that you cannot support with a graph, screenshot, page number or spreadsheet. Written docs will come back to haunt you. If you are not sure of something, then say exactly that. If your source data is suspect or incomplete – tell them and then go ahead and do your best with it.

You will want to cover the following topics in your written report:

- Load Testing Environment
    - Were the tests reproducible?
    - Was the environment stable?
    - If a cookbook were available, could the testing team deploy a default agent configuration?
    - Is the testing team interested in using APM visibility and reporting as part of their testing processes?
- Test Plan
    - If production data is available, does the test plan account for the key metrics found in production? The top 10 components in test should be the same in the top 10 components in production, for response times and invocations. Response time and volume will not always be the same (unless QA is a duplicate of production)
    - Could the test plan easily accommodate the shorter test cycles suitable for APM?
- Baselines
    - Results of Configuration Baseline and discussion
        - Was an optimal configuration identified?
        - Was custom tracing required? And is the resulting tracing confirmed safe?

        - Any unusual observations during application startup, and initial effect of load?
    - Results of Application Baseline and discussion
        - Is visibility sufficient to support triage
        - Are visibility gaps present?
    - Results of Performance Baseline and discussion
        - During steady-state and stress-to-failure, collect screenshots of the highest response time and highest invocations
    - Discussion of how key metrics were identified and thresholds determined
    - Overview of the hierarchal dashboards
    - Overview of the baseline reports
- Application Audit Findings
    - Memory management and stability
    - Response times for components that are representative of typical and problematic
    - Invocations for components that are representative of typical and problematic
    - Metrics that strongly correlate with increasing load
    - Metrics that are independent of increasing load
    - Metrics that had unexpected or anomalous behavior, even as testing was successful

FOLLOW-ON ACTIVITIES

As the intent of the Application Audit is to provide a process for tracking the performance of an application over its lifecycle, you can really expect to do multiple audits before a client is then prepared to take over the activity themselves. The ultimate goal is for the client to be self-sufficient. So while you are taking every opportunity to mentor, and thus move this activity under the client's responsibility, you will want to look for an opportunity to deliver a formal mentoring program or more simply, to assess that the client team is performing audits consistently.

REFERENCES

Artifacts

GENERIC QA Test Plan.ppt

GENERIC Introscope Agent and EM Guide.doc (environment-specific install cookbook)

App Audit Scope.doc

Presentations

QA Practice and Service Bureau with APM.ppt

APM-BP-2 Establishing a QA Practice.ppt

APM-BP Software Quality.ppt

APM Global Adoption Strategy.ppt (Using Audits, Acceptance Criteria and Pre-Production Review to standardize quality for a global organization)

PDFs

Collecting Baselines – Finding Out Which Metrics Matter.pdf

Workshops

APM BP – QA Testing and Acceptance Criteria.ppt

APM BP – Configuration Tuning – JVM and APM.ppt

APM BP – Deployment Planning.ppt (for Phased Deployment Model)

Best Practice Modules

LCMM-1 App Audit Processes

LCMM-1 Baselines

LCMM-1 Agent Customization

LCMM-1 QA Acceptance

LCMM-2 Agent Validation

LCMM-2 Alert Review and Escalation

Books

APM Best Practices – Realizing Application Performance Management ISBN 978-1-4302-3141-7

ABOUT THE AUTHOR

Michael Sydor is an Engineering Services Architect specializing in Best Practices for APM. He advises and leads client teams to establish their own APM disciplines to deliver effective triage and manage performance across the application lifecycle. Michael is also the author of “Application Performance Management – Realizing APM”, available from APress and Amazon.