Proactive Application Monitoring

Alexandre Polozoff

April 09, 2003

When applied in an appropriate context, application monitoring is more than just the data that shows how an application is performing technically. This article presents a discussion on application monitoring methods, tools, and justification, and also provides a useful overview of what metrics to collect, for which components of a Web application, and when to collect them.

Introduction

Monitoring applications to detect and respond to problems - before an end user is even aware that a problem exists - is a common systems requirement, especially for revenue-generating production environments. Most administrators understand the need for application monitoring. Infrastructure teams, in fact, typically monitor the basic health of application servers by keeping an eye on CPU utilization, throughput, memory usage and the like. However, there are many parts to an application server environment, and understanding which metrics to monitor for each of these pieces differentiates those environments that can effectively anticipate production problems from those that might get overwhelmed by them.

When applied in an appropriate context, application monitoring is more than just the data that shows how an application is performing technically. Information such as page hits, frequency and related statistics contrasted against each other can also show which applications, or portions thereof, have consistently good (or bad) performance. Management reports generated from the collected raw data can provide insights on the volume of users that pass through the application. An online store, for example, could compare the dollar volume of a particular time segment against actual page hits to expose which pages are participating in higher or lower dollar volumes.

Justifying proactive application monitoring

There are fundamentally two ways to approach problem solving in a production environment:

1. One is through continual data collection through the use of application monitoring tools that, typically, provide up-to-date performance, health and status information.
2. The other is through trial and error theorizing, often subject to whatever data is available from script files and random log parsing.

Not surprisingly, the latter approach is less efficient, but it's important to understand its other drawbacks as well. Introducing several levels of logging to provide various types of information
has long been a popular approach to in-house application monitoring, and for good reason. Logging was a very trusted methodology of the client-server era for capturing events happening on remote workstations to help determine application problems. Today, with browsers dominating the thin client realm, there is little need for collecting data on the end user's workstation. Therefore, user data is now collected at centralized server locations instead. However, with the general assumption that all possible points of logging are anticipated and appropriately coded, data collection on the server is also problematic. More often than not, logging is applied inconsistently within an application, often added only as problems are encountered and more information is needed.

In contrast, application monitoring tools offer the ability to quickly add new data - without application code changes - to information that is already being collected, as the need for different data changes with the ongoing analysis.

While logging worked well in the single user environment, there are some inherent problems with logging in the enterprise application server environment:

- Clustered environments are not conducive to centralized logs. This is a systemic problem for large environments with multiple servers and multiple instances of an application. On top of the problem of exactly how one is to administer the multiple logs, is the user's ability to bounce around application servers for applications that do not use HTTP Session objects. Coordinating and consolidating events for the same user spread across multiple logs is extremely difficult and time consuming.
- Multiple instances of applications and their threads writing to the same set of logs imposes a heavy penalty on applications that essentially spend time synchronized in some logging framework. High volume Web sites are an environment where synchronization of any kind must be avoided in order to reduce any potential bottlenecks that could result in poor response times and, subsequently, a negative end user experience.
- Different levels of logging requires additional attention: when a problem occurs, the next level of logging must be turned on. This means valuable data from the first occurrence of the problem is lost. With problems that are not readily reproducible, it's difficult to predict when logging should be on or off.
- Logs on different machines can have significant timestamp differences, making correlation of data between multiple logs nearly impossible.
- Beyond the impact of actually adding lines of code to an application for monitoring, additional development impacts include:
  - Code maintenance: The functionality, logical placement and data collected will need to be kept up, hopefully by developers who understand the impact of the code change that was introduced.
  - Inconsistent logging: Different developers may have drastically different interpretations of what data to collect and when to collect it. Such inconsistencies are not easily corrected.
  - Developer involvement: Involving developers in problem determination becomes a necessity with log-based approaches, since the developer is usually the best equipped to interpret the data.
Application monitoring accomplished through coding is rarely reused. Certainly the framework itself can be reused, but probably not the lines of code inserted to capture specific data.

When logging to a file, the impact on the server's file I/O subsystem is significant. Few things will slow down an enterprise application more than writing to a file. While server caches and other mechanisms can be configured to minimize such a hit, this is still a serious and unavoidable bottleneck, especially in high volume situations where the application is continually sending data to the log.

While Aspect-Oriented Programming is proving a valuable technology for logging, it has yet to be embraced by the technical community.

Not surprisingly, it is also common for development teams to try to collect basic performance data using their logging framework, capturing data such as servlet response time, or the timings of specific problematic methods, etc., in order to better understand how the application performs. This activity is victim to the same disadvantages mentioned above, in that any suspected problem points are correctly identified and instrumented. If new data points are identified, then the application must be modified to accommodate the additional data collection, retested and then redeployed to the production environment. Naturally, such code also requires continual maintenance for the life of the application.

**Proactive Application Monitoring Tools**

The benefits of a proactive, tool-based approach to application monitoring are many:

- **No code**
  This, by far, is the single most valuable benefit regarding a tools-based approach. Application monitoring tools, through the ability of classloader instrumentation and other Java techniques, allows for the seamless and invisible collection of data without writing a single line of code.

- **Fewer developer distractions**
  With application monitoring no longer a focal point, developers can instead concentrate on the logic of the application.

- **Non-application specific**
  Application monitoring tools are not developed for anything more specific than the Java language and WebSphere Application Server environment.

- **Reusability**
  Application monitoring tools are written to generically capture data from any application, resulting in a tremendous amount of reuse built into the tooling itself. Without doing anything extraordinary, an application monitoring tool can capture data for a variety of applications as they come online.

- **Reliability**
  While you should still perform due diligence to ensure that a tool is working properly in your environment, application monitoring tools from major vendors are generally subject to extensive testing and quality assurance for high volume environments.

- **Understandable results**
  Consolidation of data occurs at some central console, and the results can be readily understood by a systems administrator. Only when the system administrator has exhausted
all resources would developers need to assist in troubleshooting by examining data from a variety of subsystems.

- **Cost**
  Yes, there is the initial expenditure of procuring such a tool, but there is also the very real possibility of eventual cost savings - particularly in terms of time.

### Application Monitoring 101

A WebSphere Application Server-based application has, at the very least, two or more of the components identified in Figure 1:

1. servlet container
2. EJB container
3. HTTP Session objects
4. connection pool to database(s)
5. JVM memory.

Each one of these components has a variety of metrics that can be collected and monitored. When monitoring an application, specific components are identified for monitoring, depending on what it is you want to watch for, then thresholds are set to provide alerts to the team of people that can work on the particular problem. For example, if the connection pool is experiencing slower SQL timings than normal, then the back end database and network administrators would be contacted so they could figure out why this is happening.

### Figure 1. Basic components of a WebSphere Application Server environment

Application monitoring can be divided into the following categories:

1. **Fault**
   This type of monitoring is primarily to detect major errors related to one or more components. Faults can consist of errors such as the loss of network connectivity, a database server going off line, or the application suffers a Java out-of-memory situation. Faults are important events to detect in the lifetime of an application because they negatively affect the user experience.

2. **Performance**
   Performance monitoring is specifically concerned with detecting less than desirable application performance, such as degraded servlet, database or other back end resource usage.
response times. Generally, performance issues arise in an application as the user load increases. Performance problems are important events to detect in the lifetime of an application since they, like Fault events, negatively affect the user experience.

3. **Configuration**
   Configuration monitoring is a safeguard designed to ensure that configuration variables affecting the application and the back end resources remain at some predetermined configuration settings. Configurations that are incorrect, such as a too low maximum JVM heap size setting or DB2 maxapplheapsz, can negatively affect the application performance. Large environments with several machines, or environments where administration is manually performed, are candidates for mistakes and inconsistent configurations. Understanding the configuration of the applications and resources is critical for maintaining stability.

4. **Security**
   Security monitoring detects intrusion attempts by unauthorized system users.

5. **Accounting**
   Some installations charge application owners maintenance and administration fees. This type of monitoring measures usage so that, for example, organizations that have a centralized IT division with profit/loss responsibilities can appropriately bill its customers based on their usage.

Each of these five categories can also be integrated into daily or weekly management reports for the application. If multiple application monitoring tools are used, the individual subsystems should be capable of either providing or exporting the collected data in different file formats that can then be fed into a reporting tool. Some of the more powerful application monitoring tools can not only monitor a variety of individual subsystems, but can also provide some reporting or graphing capabilities.

### Historical data

One of the major side benefits of application monitoring is in being able to establish the historical trends of an application. Applications experience generational cycles, where each new version of an application may provide more functionality and/or fixes to previous versions. Proactive application monitoring provides an way to gauge whether changes to the application have affected performance and, more importantly, how. If a fix to a previous issue is showing slower response times, one has to question whether the fix provided was properly implemented. Likewise, if new features prove to be especially slower than others, one can focus the development team on understanding the differences.

Historical data is achieved by defining a baseline based upon some predefined performance test and then re-executing the performance test when new application versions are made available. This baseline has to be performed on the application at some point in time and can be superceded by a new baseline once performance goals are met. Changes to the application are then directly measured against the baseline as a measurable quantity. Performance statistics also assist in resolving misconceptions about how an application is (or has been) performing, helping to offset subjective observations not based on fact. When performance data is not collected, subjective observations often lead to erroneous conclusions about application performance.
Metrics

The following sections define a collection of metrics applicable to a typical WebSphere Application Server environment. In the vein of extreme programming, collect the bare minimum metrics and thresholds which you feel are needed for your application, selecting those that will provide the data points necessary to assist in the problem determination process. Start with methods that access backend systems and servlet/JSP response timings. Prepare to change the set of collected metrics or thresholds as your environment evolves and grows.

Keep in mind that the collection of metrics available will depend on your infrastructure. Some components, such as network switches and routers, have built-in SNMP capabilities to send traps when faults occur. Other back end resources are easily monitored by Tivoli© Distributed Monitor tools. Monitoring the application and JVM environment are available through tools such as Wily's Introscope, which is capable of emitting SNMP traps to a Tivoli console. The mix and match of tools in every environment will be different, based on technical and business requirements. What may be an effective tool in one environment may fall short in others.

Fault monitoring

Not unexpectedly, the single most comprehensive collection of metrics from the application environment is for fault monitoring. These metrics involve not only detecting application-related faults, but also those faults related to the physical server the application is running on, the back end resources being accessed, and the network connectivity components (switches, routers, etc.). Many of the metrics described in the fault grouping correlate to threshold metrics in other categories.

<table>
<thead>
<tr>
<th>Type of Monitoring</th>
<th>Applicable Metric</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware and Network</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Server availability</td>
<td>Heartbeat/ping all servers</td>
<td>UP/DOWN</td>
</tr>
<tr>
<td>Error report</td>
<td>Monitor error report logs hard errors</td>
<td>ERRORS</td>
</tr>
<tr>
<td>Network latency</td>
<td>Ping time between network components</td>
<td>UP/DOWN/SNMP traps</td>
</tr>
<tr>
<td>CPU utilization</td>
<td>CPU utilization all servers</td>
<td>&gt; 99% over x minutes</td>
</tr>
<tr>
<td>Memory utilization</td>
<td>Memory utilization all servers</td>
<td>&gt; 99% over x minutes</td>
</tr>
<tr>
<td>Paging/swapping</td>
<td>OS level metric all servers</td>
<td>In process of paging/swapping</td>
</tr>
<tr>
<td>File system</td>
<td>Available file space all servers</td>
<td>Out of space</td>
</tr>
<tr>
<td>Network components</td>
<td>Capture SNMP traps</td>
<td>UP/DOWN/ERROR</td>
</tr>
<tr>
<td><strong>WebSphere Application</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Server</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Admin server process</td>
<td>Monitor admin server process</td>
<td>UP/DOWN</td>
</tr>
<tr>
<td>Application server process</td>
<td>Monitor application server process</td>
<td>UP/DOWN</td>
</tr>
<tr>
<td>Java naming server</td>
<td>Scripts to run JNDI queries</td>
<td>UP/DOWN/ERROR</td>
</tr>
<tr>
<td>Web application</td>
<td>Running</td>
<td>STARTED/STOPPED</td>
</tr>
</tbody>
</table>
Proactive Application Monitoring

<table>
<thead>
<tr>
<th>Type of Monitoring</th>
<th>Applicable Metric</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Hardware and Network</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Network latency</td>
<td>Ping time and network bandwidth measurements</td>
<td>Timings &gt; 1000 ms or network bandwidth maxed</td>
</tr>
<tr>
<td>CPU utilization</td>
<td>CPU utilization all servers</td>
<td>&gt; 80% over x minutes</td>
</tr>
<tr>
<td>Memory utilization</td>
<td>Memory utilization all servers</td>
<td>&gt; 80% over x minutes</td>
</tr>
<tr>
<td>Paging/swapping</td>
<td>OS level metric all servers</td>
<td>In process of paging/swapping</td>
</tr>
<tr>
<td>File system</td>
<td>Available file space all servers</td>
<td>&gt; 80% used</td>
</tr>
<tr>
<td>Network components</td>
<td>Capture SNMP traps</td>
<td>Degraded counters</td>
</tr>
<tr>
<td><strong>WebSphere Application Server</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Java naming server</td>
<td>Scripts to run JNDI queries</td>
<td>Response time &gt; 3 secs</td>
</tr>
</tbody>
</table>

Note that some tools provide error messages only in log files that must be monitored.

- DB2: monitor db2diag.log
- CTG: monitor CICSCLI.LOG
- SNA: monitor sna.err
- Application log files are per application. If the environment is clustered, then the log files from each application clone must be monitored.

Performance monitoring

The metrics in the performance monitoring grouping are specific to detecting degraded behavior by any of the resources related to the application.
Metrics specific to an application can involve a number of Complex Page Requests, used to determine application performance by specific functions. Some functions may have lower thresholds than others. How often the metrics need to be collected depends on the tool and metric being collected. For example, metrics such as average servlet response time and CPU Utilization should be collected at least every minute or two, whereas complex page requests may be executed only once, every 10 to 20 minutes.

**Configuration monitoring**

The variety of back end resources that can exist in a WebSphere Application Server configuration is non-trivial. In addition to these configurations, there are also a variety of configurations specific to the application. However, configuration changes occur infrequently in the production environment, making them ideal candidates for periodic monitoring on a less frequent basis.
Attempting to take configuration snapshots with the XMLConfig tool must be handled with some foresight. XMLConfig is a performance intensive application, especially in large WebSphere® Application Server environments. Therefore, scheduling XMLConfig exports during low volume or maintenance windows is recommended.

Security monitoring

Security monitoring is concerned with the ability to detect intrusion and denial of service attacks. Security monitoring can be complex, since each network component (e.g., firewall, router, third party authentication software, etc) has its own security protocols and detection capabilities. There are a number of good authoritative references on the subject of security that can help you with specific details, such as setting the appropriate monitoring points. Due to the nature of this type of monitoring, you will want to have a third party, who is competent in security, audit your installation to make sure that your monitoring points are adequately set for comprehensive threat detection.

Accounting monitoring

In environments where it is necessary to charge application owner fees based on usage, most data for accounting can be derived from the Web server access logs (a capability of the WebSphere Site Analyzer). Applications with Java fat clients that do not communicate via a Web server may require that the application provide additional logging capabilities that allow the capture of usage data. Data mining techniques can be used by large, high volume installations, but this also requires the ability to store large amounts of data for some minimum period of time.

Conclusion

Monitoring a variety of application metrics in production can help you understand the status of the components within an application server environment, from both a current and historical perspective. As more back end resources and applications are added to the mix, you need only to instruct the application monitoring tool to collect additional metrics. With judicious planning
and the right set of data, proactive monitoring can help you quickly correct negative application performance, if not help you avoid it altogether.

Interpreting raw data within a business context can help management understand how applications are performing, since the correlation of the volume statistics with, say, total revenue may be easily produced depending on the raw data you're collecting. Understanding how a site is generating revenue can help guide future changes to an application.

Perhaps it's inevitable that some application errors will occur. At the very least, proactive monitoring provides you with the ability to detect problems as they happen, and fix them before anyone notices. If problems are going to happen, it's better that you find them before your customers do.

© Copyright IBM Corporation 2003

Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)