Examining Precise for J2EE findings

This section includes the following topics:

- How to identify performance problems
- About J2EE findings

How to identify performance problems

In most cases, a Precise for J2EE workspace displays information in the context of a specific instance and time frame. However, if you want to view findings for another time frame or instance, you can change these settings using the respective drop-down lists.

See “About the Findings area” on page 11.

How to investigate a finding

When investigating finding, it is recommended to investigate findings based on their severity.

To investigate a finding

1. In the All J2EE Instances table, select the instance you want to investigate. In the Finding Details area, the top findings for a selected instance are displayed in the Findings table of a workspace.
2. Identify the findings with the highest severity rankings (red, orange, and yellow icons where red is the highest severity and yellow the lowest) in the Findings table.
3. In the Findings table, select a finding to analyze further the problem.
4. In the selected finding (the expanded view), read the data displayed for the finding and follow any links provided to view additional information (advice) or next steps (bullets) to perform the recommendation(s) that best suit(s) your needs.

About J2EE findings

Several workspace findings exist in the Findings table to help the user.

The following is a list of current Precise for J2EE findings for an instance/method:

- Heavy Entry Point
- Frequent SLA Breaches
- Heavy Method Contribution
- Excessive CPU Usage
- Excessive Garbage Collection Time
- Memory Usage Near Maximum
- High Exceptions Rate
- Excessive Lock Time
- Slow DB Request Execution
- Slow Web service Execution
- Heavy Exit Point
- Significant JDBC Activity
- Significant External Activity
- Tuning Opportunities Detected
- Locks Detected
- Exceptions Detected
- Unbalanced Activity
- Impact on Multiple Entry Point

Heavy Entry Point

Heavy entry points may indicate a potential performance irregularity in the entry point's underlying call path. When viewing information for "All Instances" in the execution tree, a high entry point finding across multiple instances will clarify if the irregularity is a result of the entry point or if it is a result of a specific instance.

> The finding is based on the total response time values, whereas by default, the execution tree is displayed according to the average response time. As a result, the specified heavy entry point is not necessarily found towards the top of the execution tree.

Working with the finding

- Select the link in the expanded finding area. This will refresh the execution tree and analysis tabs to focus on the entry point, which will now be the selected node in the execution tree. In the new Highlights tab, examine the entry point's performance overtime activity and follow the specific findings to easily navigate to and analyze the root cause of the performance issue.

Frequent SLA Breaches

SLA thresholds are defined to help the user pinpoint HTTP entry points experiencing performance issues according to specific criteria. Frequent SLA breaches and near breaches can be caused by an underlying performance issue.

Working with the finding

To effectively locate the root cause of the performance finding, perform one of the following:

- Click on the entry point's link, and then look at the overtime SLA behavior to locate and zoom in to a specific (problematic) time frame. View the findings for that time frame and drill down until you locate the root cause.
- Select the root level of the execution tree and select the Entry Points tab. A high rate of SLA breaches across the application could result from overall resource exhaustion. Open the Memory and Statistics workspace to learn more about the environment performance issues, like high memory usage, CPU usage and so on.
- Go to **AdminPoint>Settings>SLA** to view the threshold definitions. (When you have too many SLA breaches, it may be a result of thresholds that are not defined appropriately for your application).

Heavy Method Contribution

A high work time for a specific method (reflecting the method's work time only, without the underlying call path), can indicate a performance issue within the context of that method.

In the same way, a high work time for a specific occurrence of a method invoked multiple times in the execution tree can indicate a performance issue within the context of that specific occurrence.

Working with the finding

- Examine the heaviest occurrences further by following the featured link.
  The execution tree opens to the method's heaviest call path, facilitating effective navigation to the root cause. Examine the information displayed and look at the overtime graph and findings to drill down further to find the root cause of the performance issue.

> By default, information is displayed for the heaviest method's call path. To investigate the other call paths, select them from the execution tree. (They are highlighted in bold).

Excessive CPU Usage

High CPU usage may indicate that the JVM is experiencing a lack of resources. This influences the performance of applications running on the JVM.

High CPU consumption can be directly related to your JVM's activity. However, it can also result from a general lack of free CPU resources on that server.

Before working with the finding, verify that the high CPU consumption is not caused by the above independent scenario.

Working with the finding

- Examine the CPU consumption rate by following the featured link.
  The Memory and Statistics workspace appears. If you notice a consistently high CPU rate or an increase in the CPU usage overtime, open the Activity tab, observe the performance trends and findings that can explain the high CPU usage, and drill down to locate the root cause of the performance issue. If you notice a peak in the CPU consumption, zoom in to the time frame in which the peak occurred before opening the Activity tab.

> To obtain a clearer view of your application's overall CPU consumption trends, it is recommended to increase the time frame of the displayed information.

Excessive Garbage Collection Time

The application spends too much time performing Garbage Collection activities on the specified JVM. This can result from:

- Incorrect object allocation and management like performing too many object allocations, performing too large allocations, keeping references to an object after it's no longer needed, and so on. Specifically, objects that have many references from the code and are freed only after numerous Garbage Collection cycles take more time to clean, and therefore impede the application's performance.
- The application code directly invokes Garbage Collection - it is highly recommended not to interfere with the independent Garbage Collection operation. Therefore, manual executions by application code should be eliminated.

Working with the finding

- Examine the Garbage Collection time percentage trends by following the featured link.
  The Memory and Statistics workspace appears.

> To obtain a clearer view of your application's overall Garbage Collection times trends, it is recommended to increase the time frame of the displayed information.

Memory Usage Near Maximum

Each JVM has a defined amount of maximum allowed heap size. Reaching the maximum allowed heap size can cause the JVM and the application to crash. This finding is triggered when the maximum used heap is getting dangerously close to that limit.

Working with the finding

- Examine the memory usage trends by following the featured link.
  The Memory and Statistics workspace appears. Specifically focus on the overtime graph of the maximum used heap compared to the maximum allowed heap size.
- If the maximum and/or minimum heap usage graphs indicate rising trends, consider switching on the Leak Seeker through the Settings screen.

  > This is recommended in Non-production mode only.

  > To obtain a clearer view of your application's overall memory usage trends, it is recommended to increase the time frame of the displayed information.

High Exceptions Rate

Exceptions handling has a high price in terms of processing time. It is also takes time away from a performing application's business logic. Therefore, a high exceptions rate can significantly influence application performance.

A high exceptions rate can also indicate that the application code is using exceptions as part of its natural flow, and not only as a means to handle errors and unexpected situations.

Working with the finding

- Examine the exceptions information further by following the featured link.
  The Exceptions tab opens, featuring all exceptions types and their total occurrences in the code. Examine the exceptions and their stack traces to understand where you need to focus the investigation.
- Follow the Monitored Caller links of relevant exceptions to locate the closest monitored methods inside the execution tree. You may continue further, or forward this information for next-level handling.

Excessive Lock Time

A significant percentage of the selected entity's total response time is spent waiting for lock acquisitions.

Waiting for locks means that the online activity (or batch process) is on hold, and is waiting for another activity or process to release the lock. Typically, eliminating locks can improve the performance of your transactions.

A possible solution is to consider tuning your transaction performance to reduce the time spent waiting for locks.

Working with the finding

- Examine the lock acquisitions further by following the featured link.
  The Locks tab opens, featuring all methods participating in a lock. Examine which methods are competing against each other to acquire locks. Consider whether the lock is indeed protecting a shared resource and whether access to the resource can be made more efficient, reducing the transaction's running time.

Slow DB Request Execution

A significant percentage of the selected entity's total response time is spent waiting for a specific DB request execution.

A possible solution is to tune your DB requests to optimize transaction performance.

Working with the finding

- Click on the queries to find their occurrences in the call tree, find the target DBs, and inspect their behavior over time.
- Examine the DB request further by following the featured link.
  The SQL and Exit Points tab opens, displaying details for SQL statements that the selected entity or its underlying call tree is waiting for, according to their contribution to the overall performance. From this view of the overall external activity:
  - Follow one of the slowest DB requests to drill-down within the call tree and check the load balancing information further by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.
  - If the database instance is monitored by Precise, follow the "Analyze" link in the Highlights tab to drill down to the respective expert view (for example, Precise for Oracle), and see how it could be more efficiently tuned.
- Consider parallelizing the external activity to run while processing is being done on the JVM side.

Slow Web service Execution

A significant percentage of the selected entity's total response time is spent waiting for a specific external Web service execution.

A possible solution is to tune the external Web service to optimize transaction performance.

Working with the finding

- Follow the featured link and examine which call paths invoke the external web service and which call paths are the heaviest. From this view of the overall external activity:
  - Follow one of the web service requests to drill-down within the call tree and check the load balancing information further by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.
  - If the Web service is running on a J2EE or .NET instance which is monitored by Precise, follow the "Analyze" link in the Highlights tab to drill down to the respective expert view (for example, Precise for Oracle), and see how it could be more efficiently tuned.
- Consider parallelizing the external activity to run while processing is being done on the JVM side.
- Check the load balancing information of the Web service by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.

Heavy Exit Point

A significant percentage of the selected entity's total response time is spent waiting for external activity (for example, SAP RFCs, Tuxedo service requests, RMI calls, and so on).

A possible solution is to consider tuning your transaction performance and the time spent executing external activity.

Working with the finding

- Examine the external activity further by following the featured link.
  The SQL and Exit Points tab opens, displaying details for exit points that the selected entity or its underlying call tree is waiting for, according to their contribution to the overall performance. DB activity is featured in this list as SQL statements.
  - Click on a link to examine the call paths that invoke the external activity, and focus on the heaviest.
  - If the exit point is monitored by Precise, follow the "Analyze" link in the Highlights tab to drill down to the respective expert view (for example, Precise for Oracle), and see how it could be more efficiently tuned.
- To understand the influence the exit point has on the entire application, open the Impact tab to view the impact of the exit point on all entry points and call paths in the application.
- Check the load balancing information of the exit points by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.
- Consider parallelizing the external activity to run while processing is being done on the JVM side.

Significant JDBC Activity

A significant percentage of the selected entity's total response time is spent waiting for DB activity invoked by a JDBC request.

This can result from a specific heavy statement or a collection of many, relatively short statements, being executed. A possible solution is to consider tuning your transaction performance to reduce the time spent executing DB statements.

Working with the finding

- Examine the external activity further by following the featured link.
  The SQL and Exit Points tab opens, displaying details for exit points that the selected entity or its underlying call tree is waiting for, according to their contribution to the overall performance. DB activity is featured in this list as SQL statements. From this view of the overall external activity:
  - Consider unifying queries to eliminate communication and query overheads.
  - Follow one of the heaviest SQL statements' links to drill-down within the call tree to locate the target DB and perform one of the following:
    - Check the overtime activity graph and summary area in the highlights tab.
    - Check the load balancing information further by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.
- Consider parallelizing the queries to run while processing is being done on the JVM side.
- To understand the influence the exit point has on the entire application, open the Impact tab to view the impact of the exit point on all entry points and call paths in the application.

Significant External Activity

A significant percentage of the selected entity's total response time is spent waiting for external activity (for example, SAP RFCs, Tuxedo service requests, RMI calls, and so on).

This can either indicate that a specific external activity is experiencing performance issues, or a large number of external calls, each of them relatively short, producing a high total response time.

Working with the finding

- Examine the external activity further by following the featured link.
  The SQL and Exit Points tab opens, displaying details for exit points that the selected entity or its underlying call tree is waiting for, according to their contribution to the overall performance.
  From this view of the overall external activity:
  - Consider unifying requests to eliminate communication overheads.
  - Consider using an internal cache mechanism for reducing external calls, in case the same calls are being invoked repeatedly.
- Follow one of the heaviest exit points' links to drill-down within the call tree, and perform one of the following:

    - Check the overtime activity graph and summary area in the Highlights tab.
    - Check the load balancing information further by opening the Load Balance tab. Examine whether it is relatively heavier when being called from specific JVMs.
    - To understand the influence the exit point has on the entire application, open the Impact tab to view the impact of the exit point on all entry points and call paths in the application.
    - If the exit point is monitored by Precise, follow the "Analyze" link in the Highlights tab to drill down to the respective expert view (for example, Precise for Oracle), and see how it could be more efficiently tuned.
- Consider parallelizing the external activity to run while processing is being done on the JVM side.

Tuning Opportunities Detected

The selected method showed a high work time. This may indicate that the selected method is the last branch of the call tree, or that visibility to the selected method's underlying call path can be enhanced by adding instrumentation.

Working with the finding

- If the information regarding the method and its performance metrics is sufficient, forward this information over to the Java expert or developer for next-level handling using either the email or print option.
- If more accurate pinpointing is needed, increase the level of visibility by adding instrumentation for all methods in the selected method's call tree by updating the instrumentation definitions. Following a JVM restart, you will see a detailed breakdown of the work time of the selected method and its call tree, enabling easy identification of the specific problematic method. For more information, see the About instrumenting all calls from a method section in the Precise Administration Guide.

Locks Detected

While executing the selected context and its underlying call tree, time was spent waiting for lock acquisitions.

While waiting for lock acquisition, the online activity (or batch process) is put on hold until another activity or process acquires the lock. Therefore, eliminating locks can improve the performance of your transactions.

Working with the finding

- Examine the lock details by following the featured link.
  The Locks tab opens, displaying all methods involved in a locking activity. Examine which methods are competing against each other for acquiring locks.
- To examine methods with considerable lock time, follow the lock's link to locate it within the execution tree.
  Consider whether the lock is indeed protecting a shared resource and whether access to the resource can be made more efficient, thus reducing the transactions running time.

Exceptions Detected

Exceptions were thrown in the selected context or its underlying call tree.

This may cause users (or batch activities) to experience performance or availability issues.

Working with the finding

- View the full details of the thrown exceptions by following the featured link.
  The Exceptions tab opens, displaying all thrown exceptions, their frequency, and the full stack trace. Use this information to determine whether this behavior is normal or needs further attention, or forward a link to the displayed tab to a Java expert or developer for next level handling.

Unbalanced Activity

The selected entity was invoked in multiple JVMs and encountered significantly different response times across the JVMs.

A transaction executed on a cluster of JVMs may encounter different service levels per execution, depending on the activity of the specific cluster's JVMs executing the transaction.

Working with the finding

- Examine the affected paths and transactions by following the featured link.
  The Load Balancing tab opens, displaying an overview of the selected entity's behavior across the different JVMs, and providing the ability to select the specific JVM and examine its relative performance against the application's average.
- Select a JVM row and check its overtime response time and number of executions.
    - In case of a high or growing number of executions over time, the difference in response time may be the result of load balancing issues. The affected JVMs may be getting more requests than their counterparts, and may not be able to cope with the increasing load. Check your load balancing component, as well as the scalability of the JVM's software.
    - When there are no load balancing issues, but the JVM still has a high response time, this may be a result of a resources shortage on one or more of the JVMs. Go to the Memory & Statistics workspace and examine the behavior of the affected JVMs, to determine whether a resource issue is affecting the performance of the application running on it. Typically, in case of resource shortage, more than one entry point will be affected.
    - When there are no load balancing issues and there is no resource shortage, select the slower JVMs from the Load Balancing tab, and drill down to examine the behavior of the entity on the specific JVM.

Impact on Multiple Entry Point

The selected method/SQL is invoked from multiple call paths and therefore affects the performance of the instance (JVM) in a cumulative manner.

Therefore, improving the performance of the selected method/SQL will impact more than its current call path, and may improve the performance of additional paths and transactions containing the method.

Working with the finding

- Examine the affected paths and transactions by following the featured link.
  The Impact tab opens, displaying a list of entry points and call paths affected by the selected method. Tuning methods/SQLs executions with notably high impact rates will positively affect the overall performance.
- Take note of high variations between method response times when called from different paths. Such variations may indicate a dependence of the method's performance on context. Therefore, drill down to the problematic context(s) for further tuning.