HP Performance Engineering

Best Practices Series

for Performance Engineers and Managers

Performance Monitoring Best Practices

Document Release Date: June 2012

Software Release Date: June 2012

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Third-Party Web Sites

HP provides links to external third-party Web sites to help you find supplemental information. Site content and availability may change without notice. HP makes no representations or warranties whatsoever as to site content or availability.

Copyright Notices

Trademark Notices

This document contains information from the following sources:

Microsoft® and Windows® are U.S. registered trademarks of Microsoft Corporation.

Oracle® is a registered US trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of The Open Group.

The Apache Software Foundation (http://www.apache.org/) licensed to you for use under Apache License, Version 2.0 (the "License"). You may obtain a copy of the License at http://www.apache.org/licenses/LICENSE-2.0.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to:

http://h20230.www2.hp.com/selfsolve/manuals

This site requires that you register for an HP Passport and sign-in. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support web site at:

http://www.hp.com/go/hpsoftwaresupport

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

Table of Contents

Welcome to This Guide

Welcome to *HP Performance Monitoring Best Practices*.

This guide provides concepts, guidelines, and practical examples on best implementation of performance testing monitoring in various environments.

This chapter includes:

About HP Performance Monitoring

HP is the market leader in the Automated Performance Testing. This is a discipline that leverages products, people, and processes to reduce the risks of application, upgrade, or patch deployment. At its core, automated performance testing is about applying production workloads to pre-deployment systems while simultaneously measuring system performance and end-user experience. A well-constructed performance test answers questions such as:

- Does the application respond quickly enough for the intended users?
- Will the application handle the expected user load and beyond?
- Will the application handle the number of transactions required by the business?
- Is the application stable under expected and unexpected user loads?
- Are you sure that users will have a positive experience on go-live day?

By answering these questions, automated performance testing quantifies the impact of a change in business terms. This, in turn, makes clear the risks of deployment. An effective automated performance testing process helps you make more informed release decisions, and prevents system downtime and availability problems.

HP provides two products in the area of automated performance testing - HP LoadRunner and HP Performance Center. Each focuses on different markets, but both are built on the proven and shared foundation of supported protocols, monitors, and more.

HP LoadRunner enables testing system under controlled and peak load conditions. To generate load, LoadRunner runs thousands of virtual users (Vusers) that are distributed over a network. The Vusers can run on UNIX- and Windows-based platforms. Using a minimum of hardware resources, these Vusers provide consistent, repeatable, and measurable load to exercise application under test (AUT) just as real users would.

HP Performance Center is a global cross-enterprise load testing tool that you install on your organization's own infrastructure.

- Performance Center enables managing multiple, concurrent load testing projects across different geographic locations without any need to travel between them.
- Performance Center administers all internal load testing needs.
- With Performance Center, you can manage all aspects of large-scale load testing projects, including resource allocation and scheduling, from a centralized location accessible through the Web.
- Performance Center helps streamline the testing process, reduce resource costs, and increase operating efficiency.
- Performance Center helps pinpoint performance bottlenecks.
- Performance Center enables you to determine the number of users the application under test can scale up to. (This number is the **breaking point** after which application's performance starts to degrade.) This information gives clues as to what can be done to increase the application's load capacity.

To address the needs of performance monitoring teams, and to reduce time configuring and deploying relevant monitors, we have prepared the performance monitoring guidelines contained in this guide, as well as a pre-built collection of monitors that consists of default metrics, default thresholds (where applicable) and proactive tests (where applicable). All of these have been researched using best practice data and expertise from various sources including HP's operating system administrators, HP's professional services organization, technical documentation, and books from industry experts. Monitoring system performance using these guidelines will help in identifying performance bottlenecks that lead to the root cause of problems in your systems.

The purpose of this guide is to provide easy-to-use, comprehensive performance monitoring guidelines, without the need for the Performance Center user or the IT organization to be an expert on the application.

How This Guide Is Organized

HP Performance Monitoring Best Practices contains the following sections:

Part I Introduction

Introduces performance monitoring and solutions.

Part II Operating Systems

Provides best practices for monitoring Window and UNIX operating systems.

Part III Runtime Platforms

Provides best practices for monitoring Java and .NET runtime platforms.

Part IV Web Server Monitoring

Provides best practices for monitoring Apache and IIS Web servers.

Part V Application Server Monitoring

Provides best practices for monitoring WebLogic and WebSphere application servers.

Part VI Database Resource Monitoring

Provides best practices for monitoring Oracle and MSSQL Server database resources.

Who Should Read This Guide

This guide is intended for:

- Performance Engineers
- Performance CoE Managers
- QA Managers
- QA Engineers

Additional Online Resources

Troubleshooting and Knowledge Base accesses the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:
http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport user ID, go to:
http://h20229.www2.hp.com/passport-registration.html

HP Software Web site accesses the HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. The URL for this Web site is www.hp.com/go/software.

Documentation Updates

HP Software is continually updating its product documentation with new information.

To check for recent updates, or to verify that you are using the most recent edition of a document, go to the HP Software Product Manuals Web site (http://h20230.www2.hp.com/selfsolve/manuals).

Part I

Introduction

1

Introducing Performance Monitoring

Performance monitoring is part of the broader Performance Testing discipline which deals with measuring the performance of an application under test.

In addition, performance monitoring is also useful in validating and verifying other quality attributes of the system, such as scalability, reliability, and resource usage performance.

This chapter includes:

Overview of Performance Monitoring

Performance monitoring ensures that you have up-to-date information about how your application is operating under load. By analyzing performance data for your system over a multiple loads, you can define a baseline, that is, a range of measurements that represent acceptable performance under typical operating conditions. This baseline provides a reference point which makes it easier to identify problems when they occur.

In addition, when troubleshooting system problems, performance data gives you information about the behavior of system resources at the time the problem occurs, which helps in pinpointing the cause.

Finally, monitoring application performance provides you with data to project future growth and to plan for how changes in your system configurations might affect future operations.

Performance monitoring helps identify bottlenecks and verify whether the application meets its performance objectives, by collecting metrics that characterize the application's behavior under different workload conditions (load, stress, or single user operation). These metrics should then correlate with those defined in the performance objectives. Examples of such metrics can be: response time, throughput, and resource utilization (i.e. CPU, memory, disk I/O, network bandwidth). Without a good understanding of these metrics, it is very difficult to draw the right conclusions and/or pinpoint the bottleneck when analyzing performance results. We strongly recommended that you build the expertise that enables you to conduct the right analysis.

Configuring and tuning applications for optimal performance are ongoing concerns among application developers and IT organizations. The ability to figure out *why* a particular application is running slowly is a desirable skill that is part science and part art. Whatever level of skill or artistry you possess, gathering the performance data is a necessary first step to diagnosing and resolving a wide range of problems.

Performance Terminology

Quantitative aspects of performance testing are gathered during the monitoring phase. Let's take a closer look at main terms used in performance monitoring.

Two of the most important measures of system behavior are **bandwidth** and **throughput**. Bandwidth is a measure of quantity, which is the rate at which work *can be* completed, whereas throughput measures the *actual* rate at which work requests are completed.

Throughput can vary depending on the number of users applied to the system under test. It is usually measured in terms of requests per second. In some systems, throughput may go down when there are many concurrent users, while in other systems, it remains constant under pressure but latency begins to suffer, usually due to queuing. How busy the various resources of a computer system get is known as their **utilization**.

The key measures of the time it takes to perform specific tasks are **queue time**, **service time**, and **response time**.

Service Time and Queue Time

Service time measures how long it takes to process a specific customer work request.

When a work request arrives at a busy resource and cannot be serviced immediately, the request is queued. Requests are subject to a **queue time** delay once they begin to wait in a queue before being serviced.

Response Time

Response time is the most important metric and will be used consistently throughout the book to refer to the sum of service time and queue time. It can be divided into response time at the server or client as follows:

- **Latency measured at the server.** This is the time taken by the server to complete the execution of a request. This does not take into account the client-to-server latency, which includes additional time for the request and response to cross the network.
- **Latency measured at the client.** The latency measured at the client includes the request queue, the time taken by the server to complete the execution of the request, and the network latency. Deep application usage understanding is required in order to build a proper mix of activities and their popularity among the users.

Workload Profile, Capacity, and Scalability

Another important term affecting results of performance monitoring is **workload profile** which is a mix of users performing various operations in a given application under test.

Capacity describes how much work each resource can process at its maximum level of utilization, while **scalability** is often defined as the throughput of the machine or system as a function of the total number of users requesting service.

Factors Affecting Performance

It has been known for years that although software development constantly strives towards constant improvement, it will never completely be 100% perfect. An application's performance, in turn, can only be as good as in comparison to its performance objectives.

Performance problems affect all types of systems, regardless of whether they are client/server or Web application systems. It is imperative to understand the factors affecting system performance before embarking on the task of handling them.

Generally speaking, the factors affecting performance may be divided into two large categories: **project management** oriented and **technical**.

Project Management Factors Affecting Performance

In the modern Software Development Life Cycle (SDLC), the main phases are subject to time constraints in order to address ever growing competition. This causes the following project management issues to arise:

- Shorter coding time in development may lead to a lower quality product due to a lack of concentration on performance.
- Chances of missing information due to the rapid approach may disqualify the performance objectives.
- Inconsistent internal designs may be observed after product deployment, for example, too much cluttering of objects and sequence of screen navigation.
- Higher probability of violating coding standards, resulting in unoptimized code that may consume too many resources.
- Module reuse for future projects may not be possible due to the project specific design.
- Module may not be designed for scalability.
- System may collapse due to a sudden increase in user load.

Technical Factors Affecting Performance

While project management related issues have great impact on the output, technical problems may severely affect the application's overall performance. The problems may stem from the selection of the technology platform, which may be designed for a specific purpose and does not perform well under different conditions.

Usually, however, the technical problems arise due to the developer's negligence regarding performance. A common practice among many developers is not to optimize the code at the development stage. This code may unnecessarily utilize scarce system resources such as **memory** and **processor**. Such coding practice may lead to severe performance bottlenecks such as:

- memory leaks
- array bound errors
- inefficient buffering
- too many processing cycles
- larger number of HTTP transactions
- too many file transfers between memory and disk
- inefficient session state management
- thread contention due to maximum concurrent users
- poor architecture sizing for peak load
- inefficient SQL statements
- lack of proper indexing on the database tables
- inappropriate configuration of the servers

These problems are difficult to trace once the code is packaged for deployment and require special tools and methodologies.

Another cluster of technical factors affecting performance is **security**. Performance of the application and its security are commonly at odds, since adding layers of security (SSL, private/public keys and so on) is extremely computation intensive.

Network related issues must also be taken into account, especially with regard to Web applications. They may be coming from the various sources such as:

- Older or unoptimized network infrastructure
- Slow web site connections lead to network traffic and hence poor response time
- Imbalanced load on servers affecting the performance

Performance Objectives

To successfully monitor a system under load, both the approach to monitoring performance and the monitoring itself must be relevant to the context of the performance project. Therefore the first step in monitoring as part of Performance Testing Lifecycle (PTLC) should be defining **performance objectives**. These refer to data that is collected through the process of performance testing and that is expected to have value in determining or improving the quality of the product. However, these objectives are not necessarily quantitative nor directly related to other stated performance criteria.

These objectives usually include all or some of the following characteristics:

- **Contractual.** Performance objectives are usually formally defined between the business customer and the testing entity as:
  - **mandatory.** Criteria that are absolutely non-negotiable due to legal obligations, service level agreements (SLA) or fixed business needs.
  - **negotiable.** Criteria that are desired for product release but may be modified under certain circumstances. These are typically, but not necessarily, end-user focused.

- **Precision.** The wording in which quantitative aspects of performance objectives are written:
  - **exact.** Criteria should be reached exactly as written in the objectives, for example, "50% CPU utilization."
  - **approximate.** Criteria falls within certain range or has only one limit, for example, "Memory usage per process not to cross over 50MB", "Response time of at least 90% of transaction X should be equal or less than 3 sec."
- **Boundaries.** Performance objectives frequently define certain values in regard to the application under test:
  - **target.** This is the desired value for a resource under a particular set of conditions, usually specified in terms of response times, throughput and resource utilization levels.
  - **threshold.** This represents the maximum acceptable value for resources, usually specified in terms of response times, throughput (transactions per second), and resource utilization levels.

  Performance objectives and their service attributes are derived from business requirements. Monitored metrics, captured by measuring, show the progress toward or away from performance objectives.

Performance Monitoring Guidelines

There are simple general guidelines to keep in mind when preparing for performance monitoring:

- Start from a standard sampling interval. If the problem is more specific, or if you are able to pinpoint a suspected bottleneck, then lower the time period.
- Based on the sampling interval, decide on the entire monitoring session length. Sampling at frequent intervals should only be done for shorter runs.
- Try to balance the number of objects you are monitoring and the sampling frequency, in order to keep the collected data within manageable limits.
- Pick only monitors that are relevant to the nature of the application under test in order to comprehensively cover testing scenario, while avoiding redundancy of deploying similar monitors under different names.

- Too many deployed counters may overburden analysis as well as performance overheads.
- Make sure the correct system configuration (for example, virtual memory size) is not overlooked. Although this is not exactly a part of the monitoring discipline, it may greatly affect the results of the test.
- Decide on a policy towards remote machines. Either regularly run the monitor service on each remote machine in order to collect results and then transfer results to the administrator at the end of the run by bulk, or rather continuously gather metrics and move over the network to the administrator. Choose a policy based on the application under test and the defined performance objectives.
- When setting thresholds, consider any "generic" recommendations set by hardware and/or operating system vendors (for example, Average CPU usage should be below 80% over a period of time, or disk queue length should be less than 2) as relevant for any test and application.

  This does not mean that not meeting these "generic" recommendations is always bad, but it does mean that it's always worth checking the monitoring results and load test response times with other metrics.
- Choose the parameters that will monitor the most worthwhile activity of the application and its objectives. Having too much data can overburden the analysis process.
- Monitoring goals can be achieved not only by using built-in system or application objects and counters, but also by watching application-specific logs, scripts, XML files etc.
- It may be a good idea to have a small number of basic monitors constantly running (for example, in HP SiteScope), and more detailed monitoring defined for the load testing scenario during test execution.

  Measure metrics not only under load, but also for some periods before and after the load test to allow for creating a "local baseline", and verifying that the application under test goes back to the baseline once the load test is complete.

Monitoring Misconceptions

The whole purpose of performance monitoring may be loosely defined as collecting metric data for later analysis with the ultimate goal of recognizing the root causes of bottlenecks.

While this statement is usually undisputed, there are some common misconceptions that can deviate from this goal, produce high overhead and increase costs. They are:

- **Monitoring basic infrastructure is enough.**

  Monitoring system metrics (such as CPU, memory, and disk) is important, but these metrics do not provide adequate information to truly understand whether actual users or applications are experiencing performance problems. The causes of most performance problems today are usually problems with application components, as opposed to individual pieces of hardware. As a result, system monitoring alone, while still critical, will not provide an accurate or complete picture of true application performance.

- **Monitoring processes or services for an application is enough.**

  Today's applications, whether packaged, J2EE, .NET, or customized SOA applications, are complex and span multiple systems and various technologies. In order to thoroughly understand application health, detailed component monitoring and diagnostics are required to understand the complex interactions between the various services. HP Diagnostics enables you to start with the end-user business process, then drill down into application components and system layers, thus ensuring you can achieve rapid resolution of the problems that have the greatest business impact, as well as meeting service level agreements.

- **Monitoring all of the available metrics for a system or application is the best approach.**

  Collecting too much data leads to an analysis burden that can distort the revelation of real performance problems. However 100 percent coverage is not necessary or even desirable. The famous 80/20 rule - "80 percent of problems are generally caused by 20 percent of the system's or application's components" - is true for performance monitoring as well. The solution is in knowing which systems relate to critical business functions, and which ones do not.

- **All tests can be done using the same set of metrics.**

  While some metrics would most probably remain selected for the majority of load tests, good performance monitoring includes various sets of measurements depending on the type of test to be performed.

- **Monitoring the web server is usually enough.**

  When monitoring complex modern applications, understanding its architecture is essential to getting a realistic picture of the performance cause. Standard web application deployment consists of at least a web server, an application server, and a database server, in most cases spread across multiple physical machines and even physical locations. With SOA proliferation, even more infrastructure and services may be involved in generating responses to the end user. Therefore it is very important to monitor all relevant servers - especially database machines. Sometimes it may also be necessary to monitor client workstations.

Bottlenecks and Tuning

For applications to comply with performance objectives, their performance has to be monitored continuously. By monitoring, we obtain performance data which is useful in diagnosing performance problems under production-like conditions. This data may indicate the existence of a bottleneck, that is, a situation where the performance or capacity of an entire system is severely limited by a single component.

Formally speaking, a bottleneck is located on a system's critical path and provides the lowest throughput. In client-server and especially Web based systems, there may be numerous slow points such as the CPU, memory, database, network link and so on. Some of them can be identified through monitoring the operating system’s relevant counters, while some may only be pinpointed by instrumenting the application.

HP provides a product, HP Diagnostics for J2EE/.Net, that enables IT professionals to:

- Proactively detect problems in production.
- Rapidly isolate problems to system or application tiers.
- Pinpoint root causes to specific application components.

An application may perform well in the development and QA environment, but fail to scale or may exhibit performance problems in production. It is important to understand the impact of the infrastructure in which the application runs and the behavior of the many application components as they interact under load. From the diagnostic perspective, it is important to be able to isolate the problem by tier of the application architecture, by application component, and to have progressive drill-down visibility into J2EE/.Net performance problems, the J2EE/.Net environment, and into the actual logic with sufficient detail to determine the root cause of the problems.

From the business perspective though, seeing system resources fully utilized is the intended goal - after all, all these CPU units, lots of memory and discs were paid for in order to be busy as much as possible. Therefore an informal definition of bottleneck would be the situation where a resource is fully utilized *and* there is a queue of processes/threads waiting to be served.

Distributed environments are especially vulnerable to bottlenecks due to:

- Multitude of operating systems where each of the application components may reside.
- Network configuration between the components.
- Firewalls and other security measures.
- Database malfunctioning where poor schema design, lack of proper indexing and storage partitioning may greatly slow the overall system response time.
- Ineffective thread management causing a decrease in concurrent usage.
- Unverified high number of connections.
- Fast growing number of threads due to lackluster thread pool size management.

- Database connection pool size misconfiguration.
- Unoptimized frequently used SQL statements.
- No memory tuning, both physical and shared, which is required for high volume transaction processing

As mentioned above, performance monitoring ideally leads to the identification of bottlenecks and their elimination and/or application tuning.

Another application of the 80/20 rule mentioned above is that 80% of resources are consumed by 20% of operations inside any given application. Needless to say, these most popular operations are most probably the ones causing bottlenecks. Therefore improving this 20% of the code may greatly reduce overall performance.

The process of the performance tuning is by itself partly science, partly art as it may involve intervention at the design level, compile level, assembly level, and at run time. It usually cannot be done without trade-offs - normally only one or two aspects can be addressed at the time of optimization, such as: execution time, memory usage, disk space, bandwidth, power consumption, or some other resource. For example, increased caching (and request execution time) leads to greater memory consumption, multi-processor use may complicate the source code etc.

2

HP Monitoring Solutions

HP's portfolio includes dozens of monitoring solutions for multiple purposes to address all aspects of monitoring. In the field of performance validation, HP LoadRunner and HP Performance Center integrates with two of these solutions—HP Sitescope and HP Diagnostics—to facilitate a comprehensive and complete monitoring and bottleneck analysis solution.

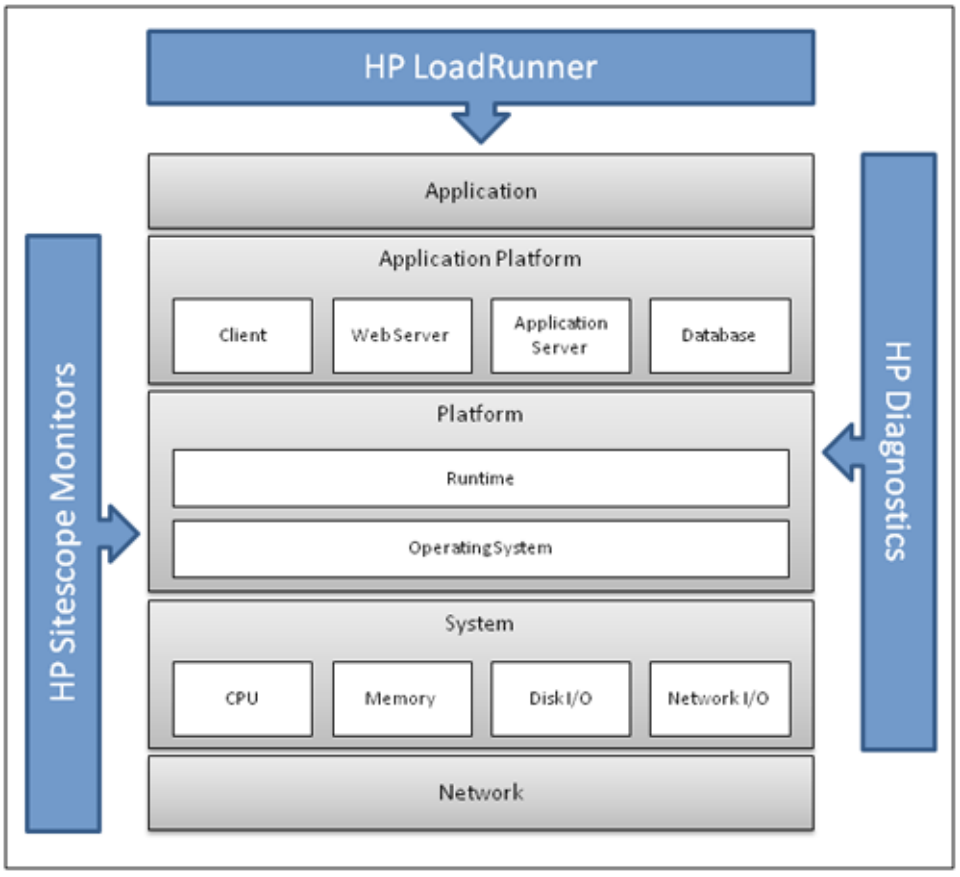
This chapter includes:

Introduction

LoadRunner and Performance Center provide a comprehensive, complete, and holistic monitoring solution when integrated with HP Sitescope and HP Diagnostics. This is done by combining the strength of each of these products:

- **LoadRunner and Performance Center.** Validate performance under load throughout simulation of typical workload and monitoring user actions in the form of transactions.
- **HP Sitescope.** Monitors the different layers of the system under test, collecting meaningful data for focusing on the bottleneck analysis process.
- **HP Diagnostics.** Isolates performance bottleneck by breaking down transaction response time into the different application layers, thus providing actionable data for problem resolution.

The following image illustrates the HP Monitoring Solution for the various layers of a System Under Test:



From a practical approach, relevant counters must be chosen for specific types of monitoring. Various metric types can be grouped under the following categories:

- **Application.** Application metrics include custom performance counters.
- **Platform.** Platform metrics are related to .NET common language runtime (CLR) on Microsoft Windows and JVM in J2EE environments. An operating system is also considered a platform.
- **System.** System metrics are related to processor, memory, disk I/O, and network I/O.
- **Network.** Network metrics are related to network bandwidth usage and latency.

For validation-oriented tests, we recommend monitoring the AUT using LoadRunner and Sitescope for identifying potential bottlenecks in transaction response time or in resource utilization. Once such a bottleneck is identified, we recommend using HP Diagnostics to isolate the issue using a more focused and shorter test, ending up with providing actionable data to the development team.

For optimization-oriented tests, we recommend involving HP Diagnostics from the beginning in order to identify potential optimization points more quickly. This approach is most suitable for tests such as stress tests, tests run against a small subsystem of the application, volume tests, and so on.

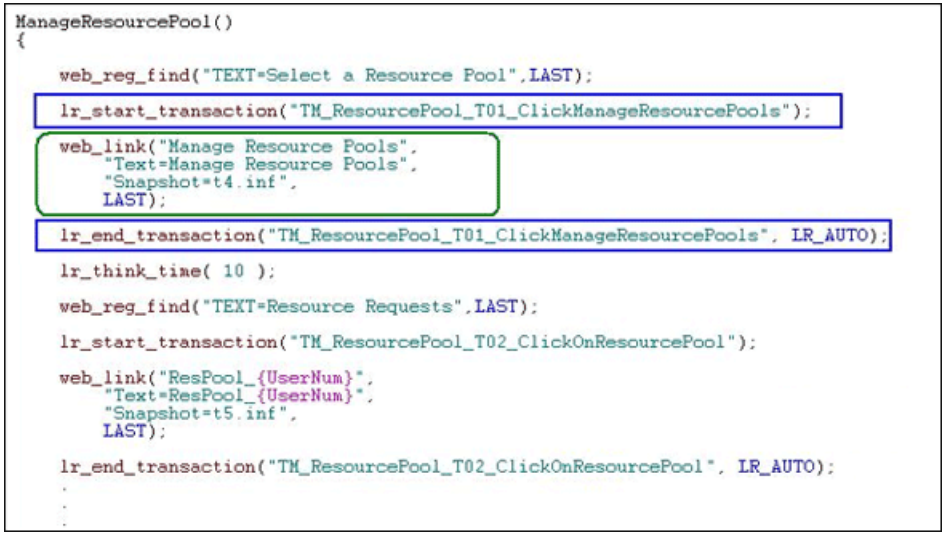
HP LoadRunner

LoadRunner and Performance Center include native monitoring capabilities that cover the immediate needs of load testing.

These include:

- **LoadRunner Data Point monitors.** Include transaction monitoring generated by VuGen scripts and automatically generated data points such as hits per seconds, throughput, and so on, when running against a Web-based application.
- **System Under Test monitors.** Include application-related metrics, such as system resource, Web server, database, and network metrics.

LoadRunner transaction monitors are the basic and most important monitors that should be applied while running a load test because they reflect the end-to-end user experience. This enables transaction validation from a business perspective, which, in turn, helps focus the testing and bottleneck analysis effort. It is recommended to use LoadRunner's Service Level Agreements to measure actual performance against performance objectives. The following image illustrates a LoadRunner script with a transaction marked to measure a web link mouse-click.



```
ManageResourcePool()
{
    web_reg_find("TEXT=Select a Resource Pool",LAST);
    lr_start_transaction("TM_ResourcePool_T01_ClickManageResourcePools");
    web_link("Manage Resource Pools",
        "Text=Manage Resource Pools",
        "Snapshot=t4.inf",
        LAST);
    lr_end_transaction("TM_ResourcePool_T01_ClickManageResourcePools", LR_AUTO);
    lr_think_time( 10 );
    web_reg_find("TEXT=Resource Requests",LAST);
    lr_start_transaction("TM_ResourcePool_T02_ClickOnResourcePool");
    web_link("ResPool_{UserNum}",
        "Text=ResPool_{UserNum}",
        "Snapshot=t5.inf",
        LAST);
    lr_end_transaction("TM_ResourcePool_T02_ClickOnResourcePool", LR_AUTO);
    .
    .
    .
```

Transaction Counters

All transaction counters are available in granularity of a single transaction and in aggregated values (totals).

Counter	Description
Transaction response time	Different response time values under different load. Average response time, maximum, percentile, and so on.
Transaction per second	Number of transactions generated per second.
Transaction success rate	Number of transactions that passed, failed, or stopped.

Web Resources Related Counters

Other data point-based monitors, provided out of the box by LoadRunner, are related to Web-based applications. These are vital counters for assessment of application ability to sustain the simulated workload.

- Hits per second
- Throughput
- HTTP responses per second
- Pages downloaded per second
- Connections
- SSL per second

LoadRunner allows generating user-defined data points from VuGen scripts. This is a very powerful tool that helps create custom, environment-specific monitors while investing only a small amount of time. This is done using VuGen's **lr_user_data_point** function; metric values can be captured from different data sources and then displayed in the LoadRunner Controller or Performance Center online graphs, as well as in LoadRunner Analysis for offline investigation and correlation with other measurements.

The following image illustrates the JBoss custom monitor. The VuGen script is configured to correlate data from the JBoss performance statistics page. The correlated values are then reported to the User-Defined Data Points graph in the Controller or on the Performance Center run page.

Lastly, as noted above, LoadRunner and Performance Center also allow monitoring of system resource utilization, databases, Web servers, application servers, and so on, using native monitors built into the products or using integration with Sitescope.

HP Sitescope

LoadRunner and Performance Center products can be configured to work together with SiteScope—the industry leading monitoring solution that can run as a standalone product or as a monitoring module for a variety of HP products such as Business Availability Center and the load testing solutions we mentioned above.

SiteScope is an agentless monitoring solution designed to ensure the availability and performance of distributed IT infrastructure, for example, servers, operating systems, network devices, applications, and application components. This Web-based infrastructure monitoring solution is lightweight, highly customizable, and does not require data collection agents to be installed on your production systems.

With SiteScope, you gain the real-time information you need to verify infrastructure operations, stay apprised of problems, and solve bottlenecks before they become critical. SiteScope also includes templates that enable development of standardized monitoring organization and speeding up of monitor deployment,. SiteScope also includes alert types that you can use to communicate and record event information in a variety of media. You can customize alert templates to meet the needs of your organization.

While native monitoring in Performance Center may cover most of an organization's average needs, it is SiteScope, with its vast collection of monitors along with pre-packaged templates, that is built to answer all possible monitoring requirements. Whether operating system measurements or application server metrics, various UNIX flavors or files inspectors—SiteScope has them all.

SiteScope was pioneered as the industry's first agentless monitoring solution. SiteScope users have benefited from its industry-proven, agentless monitoring architecture. Unlike agent-based monitoring approaches SiteScope reduces total cost of ownership by:

- Gathering detailed performance data for infrastructure components
- Reducing the time and cost of maintenance by consolidating all monitoring components to a central server
- Eliminating the possibility of an unstable agent affecting system performance

HP Diagnostics

HP Diagnostics isolates application performance problems and reduces the mean time to resolution (MTTR) of your application's performance bottlenecks. It provides actionable information to resolve performance problems.

HP Diagnostics extends LoadRunner and Performance Center to address the unique challenges of testing complicated J2EE, .NET, Enterprise Resource Planning (ERP), and Customer Relationship Management (CRM) applications across the application lifecycle.

HP Diagnostics enables you to:

- Find and solve more problems earlier in the lifecycle
- Achieve higher quality by finding the most common application problems before applications go live
- Collect concrete data to support a decision to go live with an application
- Manage and monitor applications after they have gone live with role-based visibility to solve problems quickly

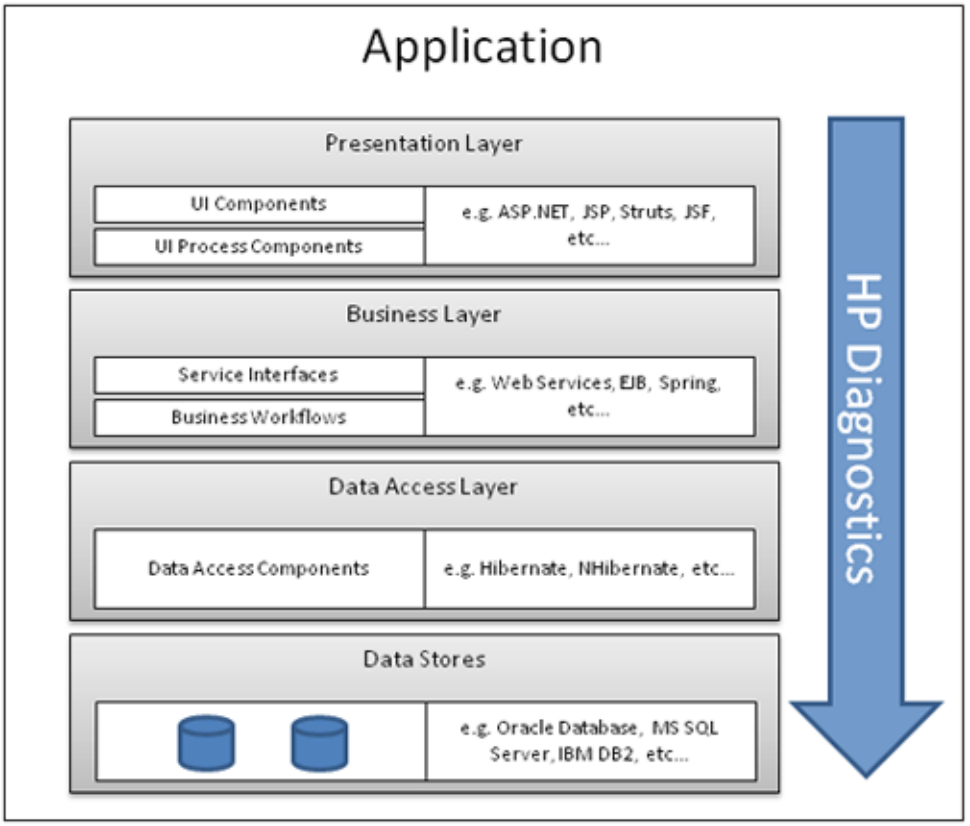
During a performance test, HP Diagnostics traces J2EE, .NET, ERP, and CRM business processes from the client side across all tiers of the infrastructure. The modules then break down each transaction response time into time spent in the various tiers and within individual components.

You gain:

- An intuitive, easy-to-use view of how individual tiers, components, memory, and SQL statements impact overall performance of a business process under load conditions. During or after a load test, you can inform the application team that the application is not scaling and provide actionable data to them.
- The ability to triage and find problems effectively with business context, enabling teams to focus on problems impacting business processes.

- The ability to more easily find components relevant to a specific business process under test. Because J2EE, ERP, and CRM applications potentially use thousands of components, this can be a challenge. HP Diagnostics software automatically detects which components are "active" when a given transaction is executed, and collects data on them for analysis. Components untouched by the business process are filtered out, letting you focus on getting the job done, rather than configuring the system.

The following diagram illustrates an example of application layers instrumented by HP Diagnostics:



Key features and benefits of HP Diagnostics:

- Drills down from slow, end-user transactions to the bottlenecked component, method or SQL statement, helping to solve memory, exception, and other common problems
- Automatically detects all components touched by a business process and traces them with no user intervention
- Provides complete application visibility across the application lifecycle, enabling higher application quality when applications go live
- Reduces mean time to resolution (MTTR) in your J2EE, .NET, ERP or CRM (Siebel, Oracle, PeopleSoft, or SAP) environment
- Integrates fully with HP Business Availability Center, LoadRunner and Performance Center

Part II

Operating Systems

3

Windows Monitoring

Performance Center provides comprehensive monitoring solutions to address load testing behavior of applications running on Windows platforms.

This chapter includes:

Overview

Since a great majority of applications used by IT organizations are Windows based, using Performance Center enables you to use Windows operating system performance counters to trace behavior of your application under test.

This chapter describes preselected collections of monitors that consist of default metrics and default thresholds (where applicable), all of which have been researched using best practice data and expertise from various sources including HP's operating system administrators, HP's Professional Services Organization, technical documentation, and books from industry experts.

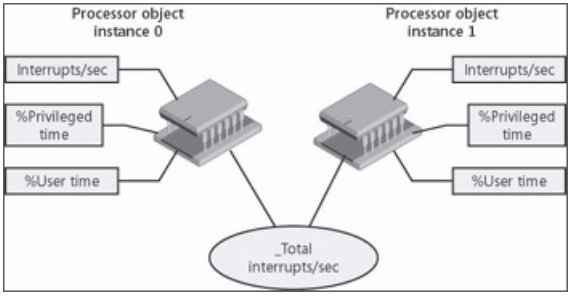
Modern Windows platforms, starting with Windows 2000 and later, provide various built-in facilities to gather, display, and reuse performance-related information. These facilities use a variety of sampling techniques to generate interval performance monitoring data that is extremely useful in diagnosing performance problems. They are designed to be efficient enough that you can run them continuously with minimal impact.

Architecture

Objects

Related performance statistics are organized into **objects**. For example, measurements related to overall processor usage, such as **Interrupts/sec** and **% User Time**, are available in the Processor object.

There might be one or more **instances** of a performance object, where each instance is named so that it is uniquely identified. For example, on a machine with more than one processor, there is more than one instance of each set of processor measurements. Each processor performance counter is associated with a specific named instance of the Processor object. The instance name is a unique identifier for the set of counters related to that instance, as shown below:



Counters

The individual performance statistics that are available for each measurement interval are numeric performance **counters**. Each performance counter you select is uniquely identified by its path, usually in the following syntax:

```
\\Computer_name\Object(Parent/Instance#Index)\Counter
```

The **Computer_name** portion of the path is optional.

For a simple object such as System or Memory that has only a single object instance associated with it, the use the following syntax:

```
\Object\Counter
```

Types of Counters

Each counter has a counter type. Knowing the counter type is useful because it indicates how the performance statistic was derived.

Some of the most important counter types are:

- **Instantaneous counters**. Display a simple numeric value of the most recent measurement
- **Interval counters.** Display an activity rate over time
- **Elapsed time counters**. Gathered on an interval basis and cannot be summarized
- **Averaging counters.** Provide average values derived for a given interval

Processor - Most Important Counters

Program execution threads consume processor (CPU) resources. These threads can be part of user-mode processes or the operating system kernel. Available performance counters measure how much CPU processing time threads and other executable units of work consume. These processor utilization measurements allow you to determine which applications are responsible for CPU consumption.

Counter	Description
% Processor Time Counter	Indicates the percentage of elapsed time that the processor spends to execute a non-idle thread
% Privileged Time Counter	Indicates the percentage of elapsed time that the process threads spent executing code in privileged mode
% Interrupt Time Counter	Indicates the time the processor spends receiving and servicing hardware interrupts during sample intervals
Processor Queue Length Counter	Indicates the number of threads in the processor queue
Context Switches Counter	Indicates the combined rate at which all processors on the computer are switched from one thread to another
System Up Time Counter	Indicates the indicator of overall system availability

% Processor Time Counter

Official Name	Processor(_Total)\% Processor Time Counter
Counter Type	Interval (% Busy)
Description	Overall average processor utilization over the interval. Every interval in which the processor is **not** running the Idle Thread, the processor is presumed to be busy on behalf of some real workload.
Usage Notes	The primary indicator of overall processor usage. Values fall within the range of 0–100% busy. The _Total instance of the processor object represents **average** total value of all the processor utilization instances.
Performance	Primary indicator to determine whether the processor is a potential bottleneck.
Operations	Sustained periods of 100% utilization might mean a runaway process. Investigate further by looking at the Process(n)\% Processor Time counter to see whether a runaway process thread is in an infinite loop.
Threshold	For response-oriented workloads, beware of sustained periods of utilization above 80–90 percent. For throughput-oriented workloads, extended periods of high utilization are seldom a concern, except as a capacity constraint.
Related Measurements	➤ Processor(_Total)\% Privileged Time (see page 49) ➤ Processor(_Total)\% User Time ➤ Processor(n)\% Processor Time ➤ Process(n)\% Processor Time Thread(n/Index#)\% Processor Time

Note: Observing heavily utilized processors on a machine does not always indicate a problem that needs to be addressed. If the other processor-related counters are increasing linearly such as **% Privileged Time** or **Processor Queue Length**, then high CPU utilization may be worth investigating.

% Privileged Time Counter

Official Name	Processor(_Total)\% Privileged Time Counter
Counter Type	Interval (% Busy)
Description	Overall average processor utilization that occurred in Privileged or Kernel mode over the interval. All operating system functions run in Privileged mode. Privileged mode includes device driver code involved in initiating device Input/Output operations and deferred procedure calls that are used to complete interrupt processing.
Usage Notes	The _Total instance of the processor object represents **average** total value of all the processor utilization instances. The ratio of **% Privileged Time** to overall **% Processor Time** (Privileged mode ratio) is workload-dependent.
Performance	Secondary indicator to determine whether operating system functions, including device driver functions, are responsible for a potential processor bottleneck.
Operations	When a runaway process thread is in an infinite loop, the state of the processor can pinpoint whether a system module is implicated in the problem.
Threshold	A figure that is consistently over 75 % indicates a bottleneck.
Related Measurements	➤ Processor(_Total)\% Interrupt Time ➤ Processor(_Total)\% DPC Time ➤ Process(n)\% Privileged Time

Note: No Privileged mode ratio is good or bad. However, a sudden change in this ratio for the same workload should trigger interest in finding out what caused the change.

% Interrupt Time Counter

Official Name	Processor(_Total)\% Interrupt Time Counter
Counter Type	Interval (% Busy).
Description	Overall average processor utilization that occurred in Interrupt mode over the interval. Only Interrupt Service Routines (ISRs), which are device driver functions, run in Interrupt mode.
Usage Notes	The _Total instance of the processor object represents **average** total value of all the processor utilization instances. Interrupt processing by ISRs is the highest priority processing that takes place. Interrupt processing is a system function with no associated process. Excessive amounts of % Interrupt Time can identify that a device is malfunctioning but cannot pinpoint which device. Use Kernrate, the kernel debugger, to determine which ISRs are being dispatched most frequently.
Performance	This counter indicates the percentage of time the processor spends receiving and servicing hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as network adapters. A dramatic increase in this counter indicates potential hardware problems.
Operations	Secondary indicator to determine whether a malfunctioning device is contributing to a potential processor bottleneck.
Threshold	Depends on the processor.
Related Measurements	➤ Processor(_Total)\Interrupts/sec ➤ Processor(_Total)\% DPC Time ➤ Processor(_Total)\% Privileged Time

Processor Queue Length Counter

Official Name	System\Processor Queue Length Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of threads that are observed as delayed in the processor Ready Queue and waiting to be scheduled for execution. Threads waiting in the processor Ready Queue are ordered by priority, with the highest priority thread scheduled to run next when the processor is idle.
Usage Notes	Many program threads are asleep in voluntary wait states. The subset of active threads sets a practical upper limit on the length of the processor queue that can be observed.
Performance	Important secondary indicator to determine whether the processor is a potential bottleneck.
Operations	An indication that a capacity constraint might be causing excessive application delays.
Threshold	On a machine with a single very busy processor, repeated observations where Processor Queue Length > 5 is a warning sign indicating that there is frequently more work available than the processor can handle readily. Ready Queue lengths > 10 are a strong indicator of a processor constraint, again when processor utilization also approaches saturation. On multiprocessors, divide the Processor Queue Length by the number of physical processors. On a multiprocessor configured using hard processor affinity to run asymmetrically, large values for Processor Queue Length can be a sign of an unbalanced configuration.
Related Measurements	Thread(parent-process\Index#)\Thread State

Context Switches Counter

Official Name	System\Context Switches/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	A context switch occurs when one running thread is replaced by another. Because Windows supports multithreaded operations, context switches are normal behavior for the system. When a User-mode thread calls any privileged operating system function, a context switch occurs between the User-mode thread and a corresponding Kernel-mode thread that performs the called function in Privileged mode.
Usage Notes	Context switching is a normal system function, and the rate of context switches that occur is a by-product of the workload. A high rate of context switches is not normally a problem indicator. Nor does it mean the machine is out of CPU capacity. Moreover, a system administrator usually can do very little about the rate that context switches occur. A large increase in the rate of context switches/sec relative to historical norms might reflect a problem, such as a malfunctioning device. Compare Context Switches/sec to the Processor(_Total)\Interrupts/sec counter with which it is normally correlated.
Performance	High rates of context switches often indicate application design problems and might also foreshadow scalability difficulties.
Operations	Context switching happens when a higher priority thread preempts a lower priority thread that is currently running or when a high priority thread blocks. High levels of context switching can occur when many threads share the same priority level. This often indicates that there are too many threads competing for the processors on the system. If you do not see much processor utilization and you see very low levels of context switching, it could indicate that threads are blocked

Threshold	Build alerts for important server machines based on extreme deviation from historical norms. As a general rule, context switching rates of less than 5,000 per second per processor are not worth worrying about. If context switching rates **exceed** 15,000 per second per processor, then there is a constraint.
Related Measurements	Thread\Context Switches/sec.

System Up Time Counter

Official Name	System\System Up Time Counter
Counter Type	Elapsed time.
Description	Shows the time, in seconds, that the computer has been operational since it was last rebooted.
Usage Notes	The primary indicator of system availability.
Performance	N/A
Operations	Reporting on system availability.
Threshold	N/A
Related Measurements	Process(n)\Elapsed Time

Note: Before measuring performance, ensure that servers and server applications are up and running and available for use.

Memory - Most Important Counters

Windows maintains physical and virtual memory. A shortage of RAM is often evident indirectly as a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator. On 32-bit systems, virtual memory is limited to 4 GB divided between 2 GB private area and 2 GB shared area. Having large amounts of physical memory does not prevent from shortage of virtual memory and may lead to fatal crashes in case of **memory leaks** when application does not release allocated memory after usage.

When observing a shortage of available RAM, it is often important to determine how the allocated physical memory is being used and count resident pages of a problematic process known as its working set.

Counter	Description
Available Bytes Counter	Indicates the amount of physical memory available to processes running on the computer
Working Set Counter	Indicates the number of resident pages of each process
Pages/sec Counter	Indicates the rate at which pages are read from or written to disk to resolve hard page faults
Page Reads/sec Counter	Indicates that the working set of the process is too large for the physical memory and that it is paging to disk
Pool Nonpaged Bytes Counter	Indicates the size of an area of system memory (physical memory used by the operating system) for objects that cannot be written to disk, but must remain in physical memory as long as they are allocated
Paged Pool Bytes Counter	Indicates memory leaks
Paged Pool Failures Counter	Indicates the number of times allocations from the paged pool have failed
Cache Bytes Counter	Indicates the size of the static files cache

Counter	Description
System Cache Resident Bytes Counter	Indicates the number of resident pages allocated to the System File Cache
Committed Bytes Counter	Indicates extreme paging leading to slow and irregular response times

Available Bytes Counter

Official Name	Memory\Available Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The set of resident pages for a process. The number of allocated pages in RAM that this process can address without causing a page fault to occur.
Usage Notes	It is calculated by adding the amount of space on the Zeroed, Free, and Standby memory lists. Free memory is ready for use; Zeroed memory are pages of memory filled with zeros to prevent later processes from seeing data used by a previous process; Standby memory is memory removed from a process' working set (its physical memory) on route to disk, but is still available to be recalled.
Performance	If memory is scarce, Process(n)\Working Set tells you how much RAM each process is using.
Operations	N/A
Threshold	A consistent value of less than 20–25% of installed RAM is an indication of insufficient memory.
Related Measurements	➤ Memory\Available Byte ➤ Memory\Committed Bytes ➤ Process(n)\Private Bytes ➤ Process(n)\Virtual Bytes ➤ Process(n)\Pool Paged Bytes

Working Set Counter

Official Name	Process(*)\Working Set Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The set of resident pages for a process. The number of allocated pages in RAM that this process can address without causing a page fault to occur.
Usage Notes	Process(n)\Working Set tracks current RAM usage by active processes. Some server applications, such as IIS, Exchange, and SQL Server, manage their own process working sets. Monitor Process(_Total)\Working Set in the Process object to see how RAM is allocated overall across all process address spaces.
Performance	If memory is scarce, Process(n)\Working Set tells you how much RAM each process is using.
Operations	N/A
Threshold	Consistent increase of 10% or more warns about limited physical memory.
Related Measurements	➤ Memory\Available Bytes ➤ Memory\Committed Bytes ➤ Process(n)\Private Bytes ➤ Process(n)\Virtual Bytes ➤ Process(n)\Pool Paged Bytes

Pages/sec Counter

Official Name	Memory\Pages/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	The number of paging operations to disk during the interval. Pages/sec is the sum of Page Reads/sec and Page Writes/sec.
Usage Notes	Page Reads/sec counters are hard page faults. A running thread has referenced a page in virtual memory that is not in the process working set. Nor is it a trimmed page marked in transition, but rather is still resident in memory. The thread is delayed for the duration of the I/O operation to fetch the page from disk. The operating system copies the page from disk to an available page in RAM and then redispatches the thread.
Performance	Primary indicator to determine whether real memory is a potential bottleneck.
Operations	Excessive paging can lead to slow and erratic response times.
Threshold	Watch out when Pages/sec exceeds 50 per paging disk.
Related Measurements	➤ Memory\Available Bytes ➤ Memory\Committed Bytes ➤ Process(n)\Working Set

---

Note: Excessive paging can usually be reduced by adding RAM. Disk bandwidth is finite. Capacity used for paging operations is unavailable for other application-oriented file operations.

Page Reads/sec Counter

Official Name	Memory\Page Reads/sec
Counter Type	Interval difference counter (rate/second).
Description	This counter indicates that the working set of the process is too large for the physical memory and that it is paging to disk. It shows the number of read operations, without regard to the number of pages retrieved in each operation. Higher values indicate a memory bottleneck.
Usage Notes	If a low rate of page-read operations coincides with high values for Physical Disk\% Disk Time and Physical Disk\Avg. Disk Queue Length, there could be a disk bottleneck. If an increase in queue length is not accompanied by a decrease in the pages-read rate, a memory shortage exists.
Performance	Primary indicator to determine whether real memory is a potential bottleneck.
Operations	Excessive paging can lead to slow and erratic response times.
Threshold	Sustained values of more than five indicate a large number of page faults for read requests.
Related Measurements	➤ Memory\Pages/sec ➤ Memory\Page Writes/sec

Pool Nonpaged Bytes Counter

Official Name	Memory\Pool Nonpaged Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Pages allocated from the Nonpaged pool are always resident in RAM.
Usage Notes	Status information about every TCP connection is stored in the Nonpaged pool. Divide by the size of a page to calculate the number of allocated pages.
Performance	If memory is scarce, Pool Nonpaged Bytes tells you how much nonpageable RAM system functions are using.
Operations	N/A
Threshold	Watch the value of Memory\Pool Nonpaged Bytes for an increase of 10 percent or more from its value at system startup. If it indeed happens, a significant memory leak is in place.
Related Measurements	➤ Pool Paged Bytes ➤ Pool Paged Resident Bytes ➤ System Cache Resident Bytes ➤ System Code Resident Bytes ➤ System Driver Resident Bytes ➤ Process(_Total)\Working Set

Paged Pool Bytes Counter

Official Name	Memory\Paged Pool Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of committed virtual memory pages in the system's Paged pool. System functions allocate virtual memory pages that are eligible to be paged out from the Paged pool. System functions that are called by processes also allocate virtual memory pages from the Paged pool.
Usage Notes	Memory\Paged Pool Bytes reports how much virtual memory is allocated in the system Paged pool. Memory\Paged Pool Resident Bytes is the current number of Paged pool pages that are resident in RAM. The remainder is paged out.
Performance	N/A
Operations	Primarily used to identify processes that are leaking memory.
Threshold	Process(n)\Paged Pool Bytes increase of more than 10% for a specific process may point to leaking memory behavior.
Related Measurements	➤ Memory\Commit Limit ➤ Memory\% Committed Bytes in Use ➤ Process(n)\Pool Paged Bytes ➤ Process(n)\Virtual Bytes

---

Note: Some outlaw processes might leak memory in the system's Paged pool. The Process(n)\Paged Pool Bytes counter helps you to identify those leaky applications.

Paged Pool Failures Counter

Official Name	Server\Paged Pool Failures Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The cumulative number of Paged pool allocation failures that the Server service experienced since being initialized.
Usage Notes	The file Server service has a number of functions that allocate virtual memory pages from the Paged pool. If a memory leak exhausts the Paged pool, the file Server service might encounter difficulty in allocating virtual memory from the Paged pool. If a call to allocate virtual memory fails, the file Server service recovers gracefully from these failures and reports on them. Because many other applications and system functions do not recover gracefully from virtual memory allocation failures, this counter can be the only reliable indicator that a memory leak caused these allocation failures.
Performance	N/A
Operations	Primarily used to identify a virtual memory shortage in the Paged pool.
Threshold	Any nonzero value of this counter indicates a bottleneck.
Related Measurements	➤ Memory\Pool Paged Bytes ➤ Memory\Commit Limit ➤ Memory\% Committed Bytes in Use ➤ Server\Pool Paged Bytes ➤ Process(n)\Pool Paged Bytes

Cache Bytes Counter

Official Name	Memory\Cache Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The set of resident pages in the system working set. The number of allocated pages in RAM that kernel threads can address without causing a page fault to occur.
Usage Notes	The system working set is subject to page replacement like any other working set.
Performance	If memory is scarce, Cache Bytes tells you how much pageable RAM system functions are using.
Operations	N/A
Threshold	N/A
Related Measurements	➤ Pool Nonpaged Bytes ➤ Pool Paged Resident Bytes ➤ System Cache Resident Bytes ➤ System Code Resident Bytes ➤ System Driver Resident Bytes ➤ Process(_Total)\Working Set

System Cache Resident Bytes Counter

Official Name	Memory\System Cache Resident Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of resident pages allocated to the System File Cache. This counter tracks the number of virtual memory pages from the File Cache that are currently resident in RAM.
Usage Notes	On file print and servers, System Cache Resident Bytes is often the largest consumer of RAM. It is part of the system's working set (Cache Bytes) and is subject to page trimming when Available Bytes becomes low.

Performance	When the System File Cache is not effective, performance of server applications that rely on the cache are impacted. These include Server, Redirector, NTFSs, and IIS.
Operations	Primarily used to identify processes that are leaking memory.
Threshold	N/A
Related Measurements	Memory\Cache Bytes

Committed Bytes Counter

Official Name	Memory\Committed Bytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of committed virtual memory pages. A committed page must be backed by a physical page in RAM or by a slot on the paging file.
Usage Notes	Committed Bytes reports how much total virtual memory process address spaces have allocated. If the Committed Bytes:RAM ratio is > 1, virtual memory exceeds the size of RAM, and some memory management will be necessary. As the Committed Bytes:RAM ratio grows above 1.5, paging to disk will usually increase up to a limit imposed by the bandwidth of the paging disks.
Performance	The Committed Bytes:RAM ratio is a secondary indicator of a real memory shortage.
Operations	Excessive paging can lead to slow and erratic response times.

Threshold	When Committed Bytes:RAM ratio exceeds 1.5, it clearly indicates real memory bottleneck.
Related Measurements	➤ Memory\Pages/sec ➤ Memory\Commit Limit ➤ Memory\% Committed Bytes in Use ➤ Memory\Pool Paged Bytes ➤ Process(n)\Private Bytes Process(n)\Virtual Bytes

Note: If the Committed Bytes:RAM ratio is close to or rises above 1.5, adding memory becomes unavoidable.

I/O - Most Important Counters

Through the I/O Manager stack, Windows maintains physical and logical disk operations. A **logical disk** represents a single file system with a unique drive letter. A **physical disk** is the internal representation of specific storage device - be it SCSI or RAID or SATA or other technology. When using complex storage systems such as array controllers or RAID, the underlying physical disk hardware characteristics are not directly visible to the operating system. These characteristics - namely, the number of disks, the speed of the disks, their seek time, rotational speed, and bit density as well as some optimization features such as on-board memory buffers - can have a major impact on performance. Advance features like memory buffers and command-queueing can boost the performance by 25–50 percent.

It is important to be proactive about disk performance because it tends to degrade rapidly, particularly when disk-paging activity occurs.

Counter	Description
Avg. Disk secs/transfer Counter	Indicates physical disk potential bottleneck
% Idle Time Counter	Indicates physical disk utilization
Disk Transfers/sec Counter	Indicates whether physical disk is a potential bottleneck
Avg. Disk Queue Length Counter	Indicates, although in conjunction with other counters, a potential bottleneck of a disk
Split IO/sec Counter	Indicates possible defragmentation
Free Megabytes Counter	Indicates logical disk space usage

Avg. Disk secs/transfer Counter

Official Name	Physical Disk(n)\Avg. Disk secs/transfer Counter
Counter Type	Average
Description	Overall average response time of physical disk requests over the interval. Avg. Disk secs/transfer includes both device service time and queue time.
Usage Notes	The primary indicator of physical disk I/O performance. Performance is dependent on the underlying disk configuration, which is transparent to the operating system. Individual disks range in performance characteristics based on seek time, rotational speed, recording density, and interface speed. More expensive, performance-oriented disks can provide 50% better performance.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Operations	Poor disk response time slows application response time.
Threshold	Depends on the underlying disk hardware, but usually should not be more than 18 milliseconds.
Related Measurements	➤ Physical Disk(n)\Disk Transfers/sec ➤ Physical Disk(n)\% Idle Time ➤ Physical Disk(n)\Current Disk Queue Length

---

Note: This counter may point to a large amount of disk fragmentation, slow disks, or disk failure. Multiply the values of the **Physical Disk\Avg. Disk sec/Transfer** and **Memory\Pages/sec** counters. If the product of these counters exceeds 0.1, paging is taking more than 10% of disk access time, so there is a need for more RAM.

% Idle Time Counter

Official Name	Physical Disk(n)\% Idle Time Counter
Counter Type	Interval (% Busy).
Description	% of time that the disk was idle during the interval. Subtract % Idle Time from 100 percent to calculate disk utilization.
Usage Notes	Derive disk utilization as follows: Physical Disk(n)\Disk utilization = 100% – Physical Disk(n)\% Idle Time. For disk arrays, divide disk utilization by the number of disks in the array to estimate individual disk utilization. Queue time can be expected to increase exponentially as disk utilization approaches 100%, assuming independent arrivals to the disk.
Performance	Primary indicator to determine whether a physical disk is overloaded and serving as a potential bottleneck.
Operations	Increased queue time contributes to poor disk response time, which slows application response time.
Threshold	Warning when % Idle Time is less than 20%.
Related Measurements	➤ Physical Disk(n)\Avg ➤ Disk secs/Transfer ➤ Physical Disk(n)\Disk Transfers/sec ➤ Physical Disk(n)\Current Disk Queue Length

Note: Calculate disk utilization, disk service time, and disk queue time to determine whether there is a poor performing disk subsystem, an overloaded disk, or both.

Disk Transfers/sec Counter

Official Name	Physical Disk(n)\Disk Transfers/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	The rate physical disk requests were completed over the interval.
Usage Notes	The primary indicator of physical disk I/O activity. Also known as the disk arrival rate. Also broken down by Reads and Writes: Physical Disk(n)\Disk Transfers/sec = Physical Disk(n)\Disk Reads/sec + Physical Disk(n)\Disk Writes/sec Used to calculate disk service time from % Idle Time by applying the Utilization Law.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Operations	Poor disk response time slows application response time.
Threshold	Depends on the underlying disk hardware.
Related Measurements	➤ Physical Disk(n)\Disk Transfers/sec ➤ Physical Disk(n)\% Idle Time ➤ Physical Disk(n)\Current Disk Queue Length

Avg. Disk Queue Length Counter

Official Name	Physical Disk(n)\Avg. Disk Queue Length Counter
Counter Type	Compound counter.
Description	The estimated average number of physical disk requests that are either in service or are waiting for service at the disk.
Usage Notes	A secondary indicator of physical disk I/O queuing that requires careful interpretation. Values of the Avg. Disk Queue Length counter should be interpreted based on an understanding of the nature of the underlying physical disk entity. What appears to the host operating system as a single physical disk might, in fact, be a collection of physical disks that appear as a single LUN. Array controllers are often used to create Virtual LUNs that are backed by multiple physical disks. With array controllers, multiple disks in the array can be performing concurrent operations. Under these circumstances, the physical disk entity should no longer be viewed as a single server. % Disk Read Time, % Disk Time, and % Disk Write Time are derived using the same formulas, except that the values they report are capped at 100%.
Performance	Secondary indicator to determine whether the disk is a potential bottleneck.
Operations	N/A
Threshold	Should not be higher than the number of spindles plus two.
Related Measurements	➤ Physical Disk(n)\% Idle Time ➤ Physical Disk(n)\Avg. Disk secs/Transfer ➤ Physical Disk(n)\Disk Transfers/sec ➤ Physical Disk(n)\Current Disk Queue Length ➤ Physical Disk(n)\% Disk Time

Split IO/sec Counter

Official Name	Physical Disk(n)\Split IO/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	The rate physical disk requests were split into multiple disk requests during the interval. Note that when a split I/O occurs, the I/O Manager measurement layers count both the original I/O request and the split I/O request as split I/Os, so the split I/O count accurately reflects the number of I/O operations initiated by the I/O Manager.
Usage Notes	A primary indicator of physical disk fragmentation. A split I/O might also result when data is requested in a size that is too large to fit into a single I/O. Split I/Os usually take longer for the disk to service, so also watch for a correlation with Physical Disk(n)\Avg. Disk secs/Transfer.
Performance	Secondary indicator that helps determine how often there is a need to run disk defragmentation software.
Operations	Poor disk response time slows application response time.
Threshold	Warning when split I/Os take more than 20% of Disk Transfers/sec.
Related Measurements	➤ Physical Disk(n)\Disk Transfers/sec ➤ Physical Disk(n)\Avg. Disk secs/Transfer ➤ Physical Disk(n)\% Idle Time

Note: Defragmenting disks on a regular basis or when the number of split I/Os is excessive normally improves disk performance, because disks are capable of processing sequential operations much faster than they process random requests.

Free Megabytes Counter

Official Name	Logical Disk(n)\Free Megabytes Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The amount of unallocated space on the logical disk, reported in megabytes. Because calculating free megabytes for very large file systems is time-consuming, the I/O Manager measurement layers recalculate the value of the counter approximately once every 5 minutes.
Usage Notes	A primary indicator of logical disk space capacity used.
Performance	N/A
Operations	Running out of space on the file system is usually catastrophic.
Threshold	Alert on this counter value or when Logical Disk(n)\% Free Space < 10 percent.
Related Measurements	Logical Disk(n)\% Free Space.

Network - Most Important Counters

Network traffic in Windows is measured at the lowest level hardware interface and at higher levels of network protocol, such as TCP/IP. Network interface statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received. Multiple instances of the Network Interface object are generated, one for every network interface chip or card that is installed. Higher level counters such as Protocol_Object\Segments Received/sec and Protocol_Object\Segments Sent/sec are available per supported protocols such as TCP, UDP, NetBEUI, NWLink IPX, NWLink NetBIOS, NWLink SPX, and more.

Counter	Description
Bytes Total/sec Counter	This indicates total throughput
Server Bytes Total/sec	This indicates overall server utilization in terms of network
Datagrams/sec Counter	This indicates IP protocol load
Connections Established Counter	This indicates TCP protocol connection success rate
Segments Received/sec Counter	This indicates number of TCP data segments received
% Interrupt Time	This indicates the time the processor spends on hardware devices interrupts, such as network card

Bytes Total/sec Counter

Official Name	Network Interface(n)\Bytes Total/sec Counter
Counter Type	Interval difference counter (rate/second)
Description	Total bytes per second transmitted and received over this interface during the interval. This is the throughput (in bytes) across this interface.
Usage Notes	The primary indicator of network interface traffic. Calculate network interface utilization: Network Interface(n)\% Busy = Network Interface(n)\Bytes Total/sec ÷ Network Interface(n)\Current Bandwidth The maximum achievable bandwidth on a switched link should be close to 90–95% of the Current Bandwidth counter.
Performance	Primary indicator to determine whether the network is a potential bottleneck.
Threshold	Warning when Total Bytes/sec exceeds 80% of line capacity.
Related Measurements	➤ Network Interface(n)\Bytes Received/sec ➤ Network Interface(n)\Bytes Sent/sec ➤ Network Interface(n)\Packets Received/sec ➤ Network Interface(n)\Packets Sent/sec ➤ Network Interface(n)\Current Bandwidth

Note: This counter helps identify whether the traffic at a specific network adapter is saturated and if there is a need to add another network adapter.

Server Bytes Total/sec

Official Name	Server\Bytes Total/sec Counter
Counter Type	Interval difference counter (rate/second)
Description	The number of bytes the server has sent to and received from the network. This value provides an overall indication of how busy the server is.
Usage Notes	This counter indicates the number of bytes sent and received over the network. Higher values indicate network bandwidth as the bottleneck. If the sum of Bytes Total/sec for all servers is roughly equal to the maximum transfer rates of your network, there is a need to segment the network
Performance	Primary indicator to determine whether the network is a potential bottleneck.
Threshold	Value should not be more than 50% of network capacity.
Related Measurements	Network Interface(n)\Bytes Received/sec

Datagrams/sec Counter

Official Name	IPvn\Datagrams/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	Total IP datagrams per second transmitted and received during the interval.
Usage Notes	The primary indicator of IP traffic.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Operations	Sudden spikes in the amount of IP traffic might indicate the presence of an intruder.
Threshold	Unexpected increase of more than 10% may indicate overload or security breach.
Related Measurements	➤ IPvn\Datagrams Received/sec ➤ IPvn\Datagrams Sent/sec ➤ Network Interface(n)\Packets/sec

Connections Established Counter

Official Name	TCPvn\Connections Established Counter
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The total number of TCP connections in the ESTABLISHED state at the end of the measurement interval.
Usage Notes	The primary indicator of TCP session connection behavior. The number of TCP connections that can be established is constrained by the size of the Nonpaged pool. When the Nonpaged pool is depleted, no new connections can be established.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Operations	Sudden spikes in the number of TCP connections might indicate a Denial of Service attack.
Threshold	Unexpected increase of more than 10% may indicate overload or security breach.
Related Measurements	➤ TCPPvn\Segments Received/sec ➤ TCPPvn\Segments Sent/sec ➤ Network Interface(n)\Packets/sec ➤ Memory\Nonpaged Pool Bytes

Segments Received/sec Counter

Official Name	TCPvn\Segments Received/sec Counter
Counter Type	Interval difference counter (rate/second).
Description	The number of TCP segments received across established connections, averaged over the measurement interval.
Usage Notes	The primary indicator of TCP network load. Calculate the average number of segments received per connection: TCPvn\Segments Received/sec ÷ TCPPvn\Connections Established/sec This can be used to forecast future load as the number of users grows.
Performance	Secondary indicator to determine whether the network is a potential bottleneck.
Operations	Sudden spikes in the amount of TCP requests received might indicate the presence of an intruder.
Threshold	Unexpected increase of more than 10% may indicate overload or security breach.
Related Measurements	➤ TCPPvn\Connections Established/sec ➤ TCPPvn\Segments Sent/sec ➤ IPvn\Datagrams Received/sec ➤ Network Interface(n)\Packets/sec

4

Monitoring UNIX

HP Performance Center provides comprehensive monitoring solutions to address load testing behavior of applications running on various UNIX platforms.

This chapter includes:

Overview

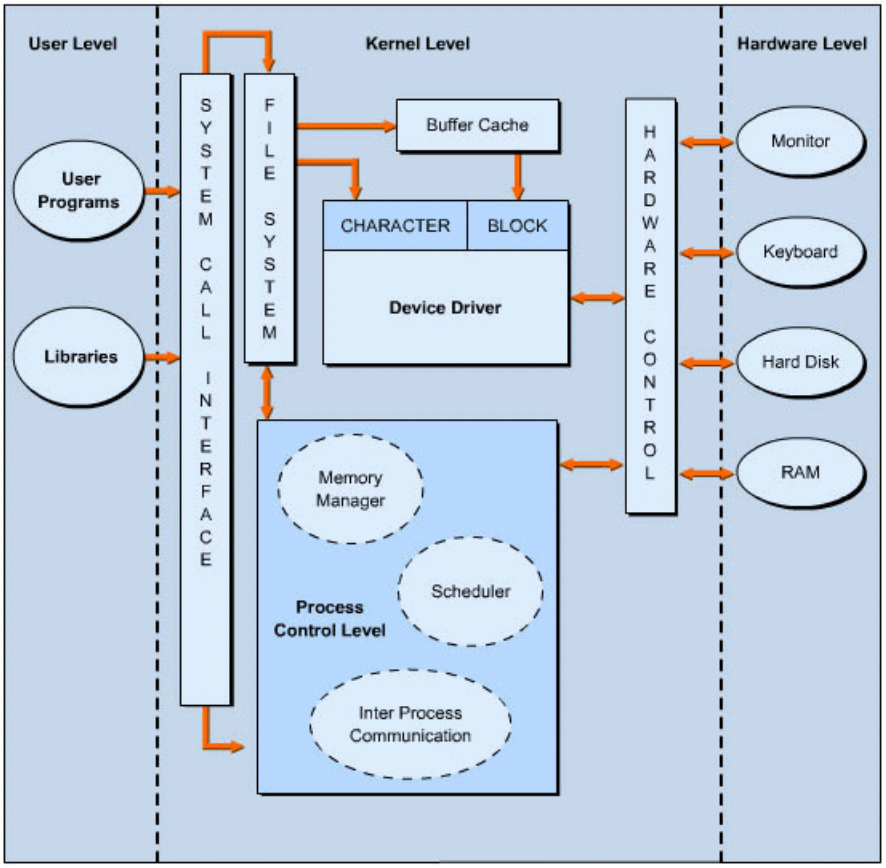
While there is an undisputed dominance of Windows based systems and applications, there are still great a deal of legacy and modern applications built on UNIX platforms. In addition to respected and well known UNIX flavors such as HP/UX, Sun Solaris, and IBM AIX, the quick expansion of Linux has caused the creation and porting of popular applications to UNIX which is known for its stability and expandability. UNIX/Linux have also became major platforms for J2EE based systems, from Apache Web servers to WebSphere application servers to Oracle database servers.

Therefore, it is no wonder that HP LoadRunner and HP Performance Center include tools to access UNIX operating system performance counters in order to trace the behavior of the application under test.

While UNIX flavors may differ on specific commands and their options, they all provide various built-in facilities to gather, display and reuse performance-related information. These facilities use a variety of sampling techniques to generate interval performance monitoring data that is extremely useful in diagnosing performance problems. They are designed to be efficient enough so that you can run them continuously with minimal impact.

Architecture

The architecture of the UNIX operating system consists of three levels: **User**, **Kernel**, and **Hardware** as shown on the image below:



The **Kernel** level is the core of UNIX and acts as an interface between the **User** and the **Hardware** levels. The **Kernel** level consists of a set of programs for various purposes. They include:

- **System call interface.** Processes and executes system calls that are functions through which a program makes a request to the operating system.

- **File system.** Coordinates with the process control and the system call interface and handles input and output of character and block data. The device driver is responsible for data I/O.

- **Process control.** Coordinates and controls the various processes in UNIX. A process is a program that is currently being executed on the operating system. That program is either a user or a system program.

- **Hardware control.** Coordinates with hardware devices, such as keyboard, monitor, hard disk, and RAM.

- **Device driver.** Communicates with system devices, such as hard disk, RAM, and printer for I/O.

Memory manager is an integral part of the UNIX architecture. It manages the amount of memory allocated to different processes running on UNIX. It is responsible for managing the memory hierarchy. A memory hierarchy consists of:

- **Buffer or cache memory.** Fast, expensive, and volatile memory with a capacity of a few kilobytes (KB) or megabytes (MB).

- **Primary memory or Random Access Memory (RAM).** Medium speed, medium price, and volatile main memory with a capacity of a few megabytes and gigabytes.

- **Secondary memory or disk storage.** Slow, cheap, and nonvolatile storage on disks and tapes with a capacity measured in gigabytes.

User programs as well as **system programs** are all termed **processes**. Their main objective is to perform a task. The system assigns a unique number called **Process Identification (PID)** to each process, and it uses these numbers to identify and manage processes. Using these numbers, the system assigns a priority to each process. After the processes are created, they may be run either in the foreground or in the background. Running a process in the background allows the system to handle multiple processes simultaneously.

Performance Resources

In UNIX there are 7 major resource types that need to be monitored and tuned:

- CPU
- Memory
- Disk space and arms
- Communications lines
- I/O Time
- Network Time
- Applications programs.

Total Execution Time

Total execution time from a user's perspective consists of wall clock time. At a process level this is measured by running the **time** command. This provides you with real time (wall clock) user code CPU and system code CPU. If user + sys > 80%, then there is a good chance the system is CPU constrained.

The components of total execution time include:

- **User-state CPU.** The actual amount of time the CPU spends running the program in the user state. It includes time spent executing library calls, but does not include time spent in the Kernel on its behalf. This value can be greatly affected by the use of optimization at compile time and by writing efficient code.

- **System-state CPU.** The amount of time the CPU spends in the system state on behalf of this program. All I/O routines require Kernel services. The programmer can affect this value by the use of blocking for I/O transfers.
- **I/O Time**. The amount of time spent servicing I/O requests.
- **Network Time.** The amount of time spent moving data.
- **Virtual Memory Performance.** Includes context switching and swapping.
- **Time spent running other programs.** When the system is not servicing this application because another application currently has the CPU.

Tools

Most UNIX flavors include built-in statistical information gathered by the operating system during process execution. Various aspects of these statistics are accessible using the following UNIX facilities:

- **rstat.** A server/daemon which returns performance statistics obtained from the Kernel
- **netstat.** Network statistics
- **nfsstat.** NFS statistics
- **time/timex.** Process CPU Utilization
- **uptime.** System Load Average
- **ps.** Process Statistics
- **iostat.** Tool for I/O
- **sar.** Bulk System Activity
- **vmstat.** Tool for Virtual Memory
- **prof.** Process Profiling
- **trace.** Used to get more depth

One of the most useful commands is **uptime**, which provides the System Load Average, although it can be used as a rough indicator only as it does not take scheduling priority into account. When **uptime** is run, it provides three load averages - the first is for the last minute, the second is for the last 5 minutes and the third is for the last 15 minutes.

The **sar** command provides a good alternative to **uptime** when used with the **-q** option. It provides statistics on the average length of the run queue, the percentage of time the run queue is occupied, the average length of the swap queue, and the percentage of time the swap queue is occupied. The run queue lists jobs that are in memory and runnable, but does not include jobs that are waiting for I/O or are sleeping. The run queue size should be less than **2**.

Note: Various UNIX flavors may include specific facilities that simplify performance monitoring. For example, Sun Solaris was enhanced with **rup** and **perfmeter** commands that are widely used instead of underlying BSD tools.

Types of Counters

Each counter has a counter type. Knowing the counter type is useful because it indicates how the performance statistic was derived. Here some most important categories of counters:

- **Instantaneous counters.** Display a simple numeric value of the most recent measurement.
- **Interval counters.** Display an activity rate over time.
- **Elapsed time counters.** Gathered on an interval basis and cannot be summarized.
- **Averaging counters.** Provide average values derived for the interval.

UNIX Monitoring with HP Tools

Unlike with Windows, performance information in UNIX is dispersed among different processes that collect various statistics. Some of these processes (daemons) are constantly running while some have to be invoked to get data.

HP LoadRunner and HP Performance Center's built-in monitoring solution for UNIX environments uses the **rstatd** daemon, which is usually already configured and running on a majority of versions. To verify whether the **rstatd** daemon is already configured, execute the **rup** command which reports various machine statistics, including **rstatd**. Using statistics collected by this daemon, the most popular counters may be obtained from the UNIX host such as CPU utilization, Context switches rate, Disk rate etc. If there is a need to get detailed view of the performance measurements, we recommended that you use the UNIX tools discussed earlier.

Instead of issuing particular commands with arguments varying between flavors, it makes great sense to deploy HP SiteScope that works in conjunction with LoadRunner and/or Performance Center.

HP SiteScope provides an adaptive infrastructure that monitors various UNIX flavors by shielding each variant specifics and grouping counters according to their purpose. This is done by configuring an adapter file to support the particular version of UNIX in need for monitoring. SiteScope uses adapter files to describe the commands that are needed to retrieve a variety of system resource information from servers running different versions of the UNIX operating system.

These commands are generic in nature, yet expand on underlying facilities of certain UNIX variants. The commands cover the wide range of the UNIX aspects and contains the following, among others:

- **disk**. Takes a disk as an argument and returns the total, free, and percentage used for the disk.
- **disks.** Returns a list of the file systems on the system.
- **memory.** The amount of used and available swap space.
- **pageFault**. The number of page faults per second. If multiple page fault lines occur, they are added up.

- **cpu.** Returns the **wait** and **idle** percentage of the CPU.
- **process.** A list of processes with long process names.

SiteScope also groups counters as per purpose (CPU, memory, I/O) as well as automatically gathers performance data in regard to instance of the group. For example, it brings CPU utilization totals along with the same data per installed processors, shows network statistics per installed network interface cards while providing totals for overall network throughput. This approach simplifies a load tester's workload because it logically merges the Windows and UNIX worlds when it is usually required to juggle between environments, sometimes even in one load test.

Processor - Most Important Counters

Every application makes use of processor (CPU) resources during execution. Requests to processor resources are divided between **user-state** and **system-state** processing.

User-state processing relates to the actual amount of time the CPU spends running the users program in the user state. It includes time spent executing library calls, but does not include time spent in the **Kernel** on its behalf.

System-state processing indicates the amount of time the CPU spends in the system state on behalf of this program. All I/O routines require **Kernel** services.

It is usually easy to recognize a CPU bottleneck: When the overall CPU utilization (average across all existing processors) is or near **100%**, and there are always processes waiting to be served. However, it is not always easy to find out *why* a CPU bottleneck occurs. Therefore it is very important to obtain prior knowledge of the application's behavior during normal times to use as a baseline when analyzing the load.

The counters below relate to system level monitoring where generic processor parameters are taken into consideration regardless of specific processes behavior.

Counter	Description
CPU Utilization	The percentage of overall time that the processor spends executing a task.
User mode CPU Utilization	The percentage of elapsed time that the processor spends executing code in user mode.
System mode CPU Utilization	The percentage of elapsed time that the processor spends executing code in system mode
Average Load	Average number of processes simultaneously in **Ready** state during the last minute.
Interrupt rate	The time the processor spends receiving and servicing hardware interruptions during sample intervals.
Context switches rate	The combined rate at which all processors on the computer are switched from one process or thread to another.

% CPU Utilization

Official Name	CPU UtilizationCounter
Counter Type	Interval (% Busy)
Description	Overall average processor utilization over the interval. Every interval in which the processor is *not* running the **Idle Thread**, the processor is presumed to be busy on behalf of some real workload. This counter is a sum of Idle + User + System utilization (names vary on different platforms). Since there is a specific Idle CPU counter on most platforms (see Related Measurements below), in order to understand overall CPU consumption, it is advisable to use the following formula: CPU Consumption = 100 - Idle CPU (%)
Usage Notes	The primary indicator of overall processor usage. Values fall within the range of 0–100% busy.
Performance	Primary indicator to determine whether the processor is a potential bottleneck.
Operations	Sustained periods of nearly 100% utilization might mean a runaway process. Usually combined with a significant Run Queue (more than 3) or processes blocked on priority (more than 3). Investigate further by looking at the User mode CPU Utilization counter to see whether it is consumed by user process or **Kernel** activities.
Threshold	For response-oriented workloads, beware of sustained periods of utilization above 80–90%.
Related Measurements	➤ CPU Utilization\%idle ➤ CPU Utilization\%usr ➤ CPU Utilization\%sys (Solaris) ➤ Processor\Idle ➤ Processor\Kernel (Linux) ➤ Processor\%idle ➤ Processor\%usr ➤ Processor\%sys (AIX)

Note: Heavy utilization of the processors on a machine does not always indicate a problem that needs to be addressed. However, should CPU Idle time drop to below 20%, it is worth investigating, and may indicate an error should it drop below 10%.

User mode CPU Utilization

Official Name	User mode CPU Utilization
Counter Type	Interval (% Busy)
Description	Overall average processor utilization that occurred in user mode over the interval, i.e. CPU was busy processing application requests.
Usage Notes	If the operating system is spending most of its time executing outside the **Kernel**, then that's typically a good thing. However, its processing power should be spent on *right* processes and no important application should be waiting to get served.
Performance	N/A
Operations	If process runs in user mode only and makes no system calls and I/O, then it may be stuck in an infinite loop. User mode processes with intensive I/O operations usually perform memory mapping. If some applications are shown as consuming all the CPU time at the expense of the application under test, the application under test would appear as being blocked on priority.
Threshold	A figure that is consistently over 50 percent indicates a bottleneck.
Related Measurements	listed in CPU utilization

System mode CPU Utilization

Official Name	System mode CPU Utilization
Counter Type	Interval (% Busy).
Description	Overall average processor utilization that occurred in system (**Kernel**) mode over the interval. All operating system functions run in **Kernel** mode. System mode includes device driver code involved in initiating device I/O operations and deferred procedure calls that are used to complete interrupt processing.
Usage Notes	In most of the cases, high system mode CPU utilization caused by other reasons. Majority of time spent by CPU in system mode occurs due to Context switching - essentially the **Kernel** running too many jobs. Another source of this would be a high Interrupt Rate (more than 30%) with underlying issues of Disk I/O or network bandwidth. Memory may be of concern too - if it is completely utilized, then swapping starts slowing the system down.
Performance	Secondary indicator to determine whether operating system functions, including device driver functions, are responsible for a potential processor bottleneck.
Operations	If no context switching or high I/O are to be blamed, then the problem lies with system calls - if it goes over 30%, use operating system tools to drill down to show stoppers.
Threshold	A figure that is consistently over 50 percent indicates a bottleneck.
Related Measurements	listed in CPU utilization

Average Load

Official Name	Average Load or Run Queue
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The number of processes that are observed as delayed in the processor Ready Queue and waiting to be scheduled for execution. Threads waiting in the processor Ready Queue are ordered by priority, with the highest priority thread scheduled to run next when the processor is idle. Number of CPU units makes no effect on the Run Queue.
Usage Notes	Many program threads are asleep in voluntary wait states. The subset of active threads sets a practical upper limit on the length of the processor queue that can be observed.
Performance	Important secondary indicator to determine whether the processor is a potential bottleneck.
Operations	An indication that a capacity constraint might be causing excessive application delays.
Threshold	On a machine with a single very busy processor, repeated observations where Average Load > 2 is a warning sign indicating that there is frequently more work available than the processor can readily handle. On multiprocessors, divide the Run Queue Length by the number of physical processors.
Related Measurements	➤ CPU Utilization ➤ Queue length\runq-sz (Solaris) ➤ Queue Statistics\runq-sz (AIX)

Interrupt Rate

Official Name	Interrupt Rate
Counter Type	Interval (% Busy)
Description	Overall average processor utilization that occurred in Interrupt mode over the interval. Only Interrupt Service Routines (ISRs), which are device driver functions, run in Interrupt mode.
Usage Notes	Interrupt processing by ISRs is the highest priority processing that takes place. Interrupt processing is a system function with no associated process. Excessive amounts of Interrupt Rate can identify that a device is malfunctioning but cannot pinpoint which device.
Performance	Indicates the percentage of time the processor spends receiving and servicing hardware interrupts. This value is an indirect indicator of the activity of devices that generate interrupts, such as network adapters. A dramatic increase in this counter indicates potential hardware problems.
Operations	Secondary indicator to determine whether a malfunctioning device is contributing to a potential processor bottleneck.
Threshold	Start paying attention when this counter goes over 30%.
Related Measurements	System mode CPU Utilization

Context Switches Rate

Official Name	Context Switches Rate
Counter Type	Interval difference counter (rate/second)
Description	A context switch occurs when one running thread is replaced by another. Because UNIX supports multithreaded operations, context switches are normal behavior for the system. When a User-mode thread calls any privileged operating system function, a context switch occurs between the User-mode thread and a corresponding Kernel-mode thread that performs the called function in System mode.
Usage Notes	Context switching is a normal system function, and the rate of context switches that occur is a by-product of the workload. A high rate of context switches is not normally a problem indicator. Nor does it mean the machine is out of CPU capacity. Moreover, a system administrator usually can do very little about the rate that context switches occur, unless there are some specific system configuration parameters to tune such as increasing amount of time each process can hold CPU by default. A large increase in the rate of context switches/sec relative to historical norms might reflect a problem, such as a malfunctioning device.
Performance	High rates of context switches often indicate application design problems and might also foreshadow scalability difficulties.
Operations	Context switching happens when a higher priority thread preempts a lower priority thread that is currently running or when a high priority thread blocks. In most of the cases, this is caused by processes created and completed very often - for example, login using shell commands. This indicates that there are too many threads competing for the processors on the system. If you do not see much processor utilization and you see very low levels of context switching, it could indicate that threads are blocked.

Threshold	N/A
Related Measurements	N/A

Note: Servers and server applications have to be up and running and available for use before measuring performance.

Processes Monitoring

UNIX is a powerful and very flexible operating system. It allows users to run processes as needed, either in the foreground or in the background. Programs running in the foreground have full read and write access, while those running in the background don't have any read access.

Performance counters are available that measure how much CPU processing time specific threads and other executable units of work consume. These processor utilization measurements allow you to determine which applications are responsible for CPU consumption.

While there is no generic facility available on all UNIX flavors, using HP SiteScope's Process object gives statistical information per selected process/thread where the following data is available (not all counters are available on all variants):

- **CPU.** CPU utilization per selected process in percentage points of overall CPU usage.
- **MEMSIZE.** Amount of memory consumed by the selected process.
- **PID.** Process ID as registered with the operating system.
- **THREADS.** Number of threads forked by the selected process.
- **USER.** Number of user sessions.

If HP SiteScope does not provide satisfactory details of process monitoring, there is always a possibility to issue built-in UNIX commands:

- **ps.** Shows a static list of currently running processes. In addition, the **ps** command shows specific details of processes, such as PID, memory used, and the command line used to run the processes. In most of the cases, adding **-aux** attribute is recommended as it gives data on user and non-terminal processes
- **top.** Shows a list of all currently running processes and the amount of memory occupied by them. The top command automatically updates the list every few seconds to display active processes on the computer.
- **proc tools.** Enables getting even more information about processes. These tools should be used with caution because they suspend the execution of processes when executed. Proc tools are located in **/var/proc** and contain **pfiles** (active processes), **pflags** (the status information and flags for processes), **pldd** (all dynamic library files attached to each process), **pmap** (address space map for processes), **psig** (actions taken for various signals and thread handlers), **prun** (runs or begins a process), **pstack** (stack trace), **pstop** (suspends the execution of a specific process).

Memory - Most Important Counters

UNIX maintains physical (resident) and virtual memory. Operating systems shield the actual amount of memory on hand from applications - hence they tend to overstate its availability. UNIX uses the term **virtual memory** which essentially includes the amount of memory allocated by programs for all their data, including shared memory, heap space, program text, shared libraries, and memory-mapped files. The total amount of virtual memory allocated to all processes on the system roughly translates to the amount of *swap* space that will be reserved (with the exception of program text). Virtual memory actually has little to do with how much actual physical memory is allocated, because not all data mapped into virtual memory will be active ('Resident') in physical memory. When the program gets an "out of memory" error, it typically means it is out of reservable swap space (Virtual memory), not out of physical (Resident) memory.

A shortage of RAM is often indirect evidence of a disk performance problem, when excessive paging to disk consumes too much of the available disk bandwidth. Consequently, paging rates to disk are an important memory performance indicator.

It is commonly said that memory today is relatively cheap - hence buying more memory can solve all problems. However, having large amounts of physical memory does not prevent a shortage of virtual memory and may lead to fatal crashes in case of memory leaks when the application does not release allocated memory after usage. In some cases, if the underlying UNIX system is set to host a database or similar high volume transaction processing application, adding a lot of memory may significantly improve database performance by allowing a larger in-memory cache.

When observing a shortage of available RAM, it is often important to determine how the allocated physical memory is being used and count resident pages of a problematic process known as its resident memory set.

In addition to the common counters below, it is important to track the usage of **cached** and **buffered** memory - a decline in amount of available free memory does not necessarily indicate a memory leak as it becomes part of it (see **%rcache/%wcache** and **bread/s bwrit/s** on Solaris and **HP/UX** and **Cached** and **Buffers** on Linux).

Tip: We recommended that you start paying attention to the memory usage when:

- There is a constant rise of overall swap usage in the system over period of time
- Memory consumption may be calculated according to the formula:

```
Used memory = All memory - (Cached + Buffered + Swap)
```

- A specific process causes constant rise of reservable swap space - in most cases, it is a clear indication of memory leak by this process.

Counter	Description
Percent Used	Indicates the total physical memory usage available to processes running on the computer.
MB Free	Indicates the total amount of memory available to running processes.
Paging Rate	Indicates the rate at which pages are read from or written to disk to resolve hard page faults, per second.
Page-in Rate	Indicates the number of pages read to physical memory, per second.
Page-out Rate	Indicates the number of pages written to pagefile(s) and removed from physical memory, per second.

Percent Used

Official Name	Percent Used
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The amount of allocated pages in RAM that can be addressed without causing a page fault to occur, in percentage point relative to all installed memory.
Usage Notes	Primary indicator of memory usage.
Performance	N/A
Operations	N/A
Threshold	A consistent value of more than 80 percent of installed RAM is an indication of insufficient memory. Watch out when it reaches 90 percent as it may fail running processes.
Related Measurements	Memory\freemem - in bytes (Solaris)

MB Free

Official Name	MB Free
Counter Type	Instantaneous (sampled once during each measurement period)
Description	Total number of megabytes of virtual memory free.
Usage Notes	Shows how much memory is available for running processes.
Performance	N/A
Operations	N/A
Threshold	N/A
Related Measurements	Memory\swap_free and Memory\swap_avail - in bytes (Solaris)

Paging Rate

Official Name	Paging Rate or Pages/sec
Counter Type	Interval difference counter (rate/second)
Description	The number of paging operations to disk during the interval. Pages/sec is the sum of Page-in/sec and Page-out/sec.
Usage Notes	When a program touches a virtual address on a page that is not in physical memory, the result will be a "page-in". When UNIX needs to make room in physical memory or when a memory-mapped file is posted, the result is called "page-out". During page-out, the whole resident memory sets are transferred to disk swap areas. In case of page-outs, the process is taken out of run queue so it gets no CPU.
Performance	Primary indicator to determine whether real memory is a potential bottleneck. Usually, there is no need to closely monitor page-ins but rather page-outs as they often point to memory bottleneck. Another source of high paging rate may be overly large file system cache buffer.
Operations	Excessive paging can lead to slow and erratic response times.
Threshold	Watch out when Paging Rate exceeds 50 per swap device.
Related Measurements	➤ Page-in Rate ➤ Page-out Rate

Notes:

- Excessive paging can usually be reduced by adding RAM. Disk bandwidth is finite. Capacity used for paging operations is unavailable for other application-oriented file operations.
- When computing swap size, it is recommended to have at least as much "reservable" swap as any application will ever request.

Page-in Rate

Official Name	Page-in Rate
Counter Type	Interval difference counter (rate/second)
Description	This counter indicates that part of the memory the process needs to access is in virtual memory and needs to be read into the physical memory for execution. It shows the number of read operations, without regard to the number of pages retrieved in each operation. Higher values indicate a memory bottleneck.
Usage Notes	This counter is of lesser importance than corresponding Page-out counter. Unless rises unexpectedly, no special attention have to be paid all the time.
Performance	Secondary indicator to determine whether real memory is a potential bottleneck.
Operations	Excessive paging can lead to slow and erratic response times.
Threshold	N/A
Related Measurements	Paging Rate

Page-out Rate

Official Name	Page-out Rate
Counter Type	Interval difference counter (rate/second)
Description	This counter indicates that the resident memory set of the process is too large for the physical memory and that it is paging to disk. It shows the number of read operations, without regard to the number of pages retrieved in each operation. Higher values indicate a memory bottleneck.
Usage Notes	If a low rate of page-out operations coincides with high values for physical disk activity, there could be a disk bottleneck. If an increase in queue length is not accompanied by a decrease in the page-out rate, a memory shortage exists.
Performance	Primary indicator to determine whether real memory is a potential bottleneck.
Operations	Excessive paging can lead to slow and erratic response times.
Threshold	Watch out when Paging Rate exceeds 50 per swap device.
Related Measurements	Paging Rate

I/O - Most Important Counters

Through I/O Manager stack, UNIX maintains physical and logical disk operations. A **logical volume** represents a single file system with a unique drive letter. A **physical (raw) volume** is the internal representation of a specific storage device - be it SCSI or RAID or SATA or other technology.

When using complex storage systems such as array controllers or RAID, the underlying physical disk hardware characteristics are not directly visible to the operating system. These characteristics - namely, the number of disks, the speed of the disks, their seek time, rotational speed, and bit density as well as some optimization features such as on-board memory buffers - can have a major impact on performance. Advance features like memory buffers and command-queueing can boost the performance by 25–50 percent.

It is important to be proactive about disk performance because it tends to degrade rapidly, particularly when disk-paging activity occurs.

Notes:

- In general, it is better to have many smaller disks than few bigger ones as this gives more flexibility to move things around and relieve I/O bottlenecks. Try splitting heavily used logical volumes across several different disks and I/O channels.
- When determining a directory path for applications, keep number of levels from the file system root to a minimum. Extremely deep directory trees may impact performance by requiring more lookups to access files. On the contrary, file access can be slowed when there are too many files (multiple thousands) in a given directory.

Transaction-oriented applications with a lot of I/O activity perform better when using raw devices instead of file system. This is usually a recommendation by most database vendors like Oracle. However, recent improvements in logical volume management brings file system devices to the level of raw volumes. In any case, it is a good idea to assign independent applications to unique physical disks to reduce possible impact on each other.

Counter	Description
%Used	Indicates relative amount of space used on each mounted file system.
Free	Indicates number of bytes free on each mounted file system.
Disk Rate	Indicates whether physical disk is a potential bottleneck.

%Used

Official Name	Filesystsems(n)\%Used
Counter Type	Interval (%)
Description	Current file system disk utilization in percentage points of full capacity.
Usage Notes	The primary indicator of physical disk I/O performance. Performance is dependent on the underlying disk configuration, which is transparent to the operating system. Individual disks range in performance characteristics based on seek time, rotational speed, recording density, and interface speed. More expensive, performance-oriented disks can provide 50 percent better performance.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Operations	Poor disk response time slows application response time.
Threshold	If this metric reaches 90%, it is an indication of warning, getting over 95% points to errors.
Related Measurements	➤ Filesystsems(n)\Use% (Linux) ➤ Filesystsems(n)\used - in bytes (Solaris)

Free

Official Name	Filesystsems(n)\Free
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The amount of unallocated space on the logical disk, reported in bytes. Because calculating free megabytes for very large file systems is time-consuming, the I/O Management measurement layers recalculate the value of the counter approximately once every 5 minutes. Main metric for planning disk usage. If no disk capacity counter is available (some UNIX flavors do supply this), using this metric and knowing overall disk volume, it is possible to calculate utilization.
Usage Notes	A primary indicator of logical disk space capacity used.
Performance	N/A
Operations	Running out of space on the file system is usually catastrophic.
Threshold	Not Available
Related Measurements	➤ Filesystsems(n)\Available ➤ Filesystsems(n)\Used (Linux) ➤ Filesystsems(n)\avail ➤ Filesystsems(n)\used ➤ Filesystsems(n)\capacity (Solaris)

Disk Rate

Official Name	Filesystsems(n)\Disk Rate
Counter Type	Interval difference counter (rate/second)
Description	The rate physical disk requests were completed over the interval.
Usage Notes	The primary indicator of physical disk I/O activity. Also known as the disk arrival rate.
Performance	Primary indicator to determine whether the disk is a potential bottleneck.
Operations	Poor disk response time slows application response time.
Threshold	Depends on the underlying disk hardware.
Related Measurements	N/A

Tips: General tips on improving I/O throughput include:

- Spreading disk I/O as much as possible - having 10 disks 10% busy is better than one disk 100% busy.
- Avoiding excessive logging - some applications allow control of log verbosity levels.
- Tuning SCSI devices - it sometimes possible to adjust maximum queue length for particular device. This usually increases parallelism at the possible expense of overloading hardware.

Notes: Some facts regarding disks:

- The smaller the I/O, the shorter the service time. The longer the I/O, the longer the service time.
- Sequential I/O is faster than random - due to decreased head movement.
- Larger I/O sizes allow maximum throughput for sequential I/O.
- Crossing various system boundaries such as file system block, buffer chain or file extent may result in breaking up one I/O request into smaller ones.
- If the busiest disk is a swap device, then most probably there is a memory bottleneck masquerading as a disk problem - you need to address the memory issue first.

Network - Most Important Counters

Networking performance has become ever more important today with proliferation of distributed and cloud applications. However, UNIX operating system usually provide limited statistics on various levels: At the lowest level hardware interface, and at higher level of network protocol such as TCP/IP. Network interface statistics are gathered by software embedded in the network interface driver layer. This software counts the number of packets that are sent and received.

Network statistics are gathered through UNIX facilities such as **netstat**, **netperf** and **iozone** and **nfsstat** (for NFS monitoring) - one for every network interface chip or card that is installed. HP products like Network Node Manager and SiteScope can collect statistics over time to give insight into the real causes of performance bottlenecks.

Networking bottlenecks are tricky to catch and analyze. Packet rates, collision rates and error rates do not always point to the cause of the problem:

- Only excessive collision rates may indicate network bottleneck. If their level is relatively low over time, it is usually normal behavior. Collisions which are essentially errors happen as a result of mismatches in either duplex or speed settings. When corrected, collision rates go down along with performance improvement.
- Sudden increase in packet rates along with high network output queue can also be an indication of network bottleneck. However, to reach informed decision, there is a need to observe pattern behavior over time.
- If NFS is extensively used, there is a need to watch data collected by **nfsstat** , especially on the server side. If NFS statistics show a lot of activity caused by one specific client, it is recommended to run the tool on that client host to identify the process.
- There can be a network bottleneck in a situation of high System-mode CPU utilization or Interrupt Rate on one of the processors while other(s) are mostly idle. Checking device configuration and hardware may be the reason.

Counter	Description
Incoming packets rate	Indicates number of Ethernet packets coming to NIC, per second.
Outgoing packets rate	Indicates number of Ethernet packets sent by NIC, per second.
Incoming packets error rate	Indicates number errors in Ethernet packets coming to NIC, per second.
Outgoing packets error rate	Indicates number of errors in Ethernet packets sent by NIC, per second.
Collision rate	Indicates number of network collisions.

Incoming Packets Rate

Official Name	Incoming Packets Rate
Counter Type	Interval difference counter (rate/second)
Description	Total bytes per second received over this interface during the interval.
Usage Notes	Primary indicator of network interface traffic - along with Outgoing Packets Rate.
Performance	Primary indicator to determine whether the network is a potential bottleneck.
Threshold	Warning when Incoming Packets Rate exceeds 40 percent of line capacity.
Related Measurements	Outgoing Packets Rate

Outgoing Packets Rate

Official Name	Outgoing Packets Rate
Counter Type	Interval difference counter (rate/second)
Description	Total bytes per second sent out over this interface during the interval.
Usage Notes	The primary indicator of network interface traffic - along with Incoming Packets Rate.
Performance	Primary indicator to determine whether the network is a potential bottleneck.
Threshold	Warning when Incoming Packets Rate exceeds 40 percent of line capacity.
Related Measurements	Incoming Packets Rate

Note: These above two counters show throughput (in bytes) across this interface, and help identify whether traffic at specific network adapters is saturated and if there is a need to add another network adapter.

Incoming Packets Error Rate

Official Name	Incoming Packets Error Rate
Counter Type	Interval difference counter (rate/second)
Description	Number of errors per second received over this interface during the interval.
Usage Notes	One of the important secondary indicators of network interface traffic - along with Outgoing Packets Error Rate.
Performance	Secondary indicator to determine whether the network is a potential bottleneck - usually a result of mismatch duplex and speed configuration.
Threshold	Warning when Incoming Packets Error Rate exceeds 0.025 errors per second.
Related Measures	Outgoing Packets Error Rate

Outgoing Packets Error Rate

Official Name	Outgoing Packets Error Rate
Counter Type	Interval difference counter (rate/second)
Description	Number of errors per second sent out over this interface during the interval.
Usage Notes	One of the important secondary indicators of network interface traffic - along with Incoming Packets Error Rate.
Performance	Secondary indicator to determine whether the network is a potential bottleneck - usually a result of mismatch duplex and speed configuration.
Threshold	Warning when Outgoing Packets Error Rate exceeds 0.025 errors per second.
Related Measures	Incoming Packets Error Rate

Tip: These above two counters track networking quality. If rates go over the designated threshold, it may be the time to take a look at the network hardware equipment.

Collision Rate

Official Name	Collision Rate
Counter Type	Interval difference counter (rate/second)
Description	The number of errors happening on the interface per second.
Usage Notes	This counter indicates the number of errors when sending or receiving data over the network. Higher values indicate network bandwidth as the bottleneck. Usually caused by hardware compression problems or bad physical connector/terminator.
Performance	Primary indicator to determine whether the network is a potential bottleneck. If values go higher than threshold, it may be a time to reevaluate network topology as network is overloaded on the segment.
Threshold	Value should not be more than 10 percent.
Related Measurements	N/A

Part III

Runtime Platforms

5

Runtime Platform Monitoring

This chapter provides an overview about runtime platform monitoring and describes the required Java and .NET application architecture.

This chapter includes:

Overview

Applications are usually developed to run on a specific operating system and their performance depends on factors that govern that operating system. Each operating system has its own set of performance parameters to monitor and tune for better performance.

Performance of applications also depends on the architectural level monitoring and tuning. However, architectural design is built upon specific technology. Therefore technology level monitoring and tuning must be addressed for better results. To achieve all these, proper guidelines must be enforced at various stages for monitoring and tuning.

While there is a multitude of technologies—general purpose and proprietary—nowadays, enterprise applications are created using either Java 2 Enterprise Edition (J2EE) or its Microsoft counterpart, the .NET Framework. Developers can now build business solutions in less time and with more functionality and robustness than ever before.

Designing these solutions is not necessarily straightforward, and with more features and functionality, the resulting product can potentially be of poor quality. An application may perform well in the development and QA environment, but fail to scale or may exhibit performance problems in production.

It is important to understand the impact of the infrastructure in which the application runs and the behavior of the many application components as they interact under load.

The deployment lifecycle for many web-facing J2EE and .NET applications is compressed, due to increased pressure for quick time-to-market. Boundaries between development, QA, deployment, and production stages and IT groups are blurred. Centralized IT organizations may be managing hundreds of applications, with little depth of each. IT staff skills for J2EE may not be developed enough.

Many applications have not been sufficiently designed for performance and scalability, with thorough consideration of design and usage patterns, and adequate attention to planning and testing performance against well-defined service objectives. J2EE scalability capabilities, although extensive, do not substitute for such efforts. The same goes for .NET configuration settings—for example, buffering, session timeout, application protection levels and logging configuration can impact your .NET application performance under load.

Architecture

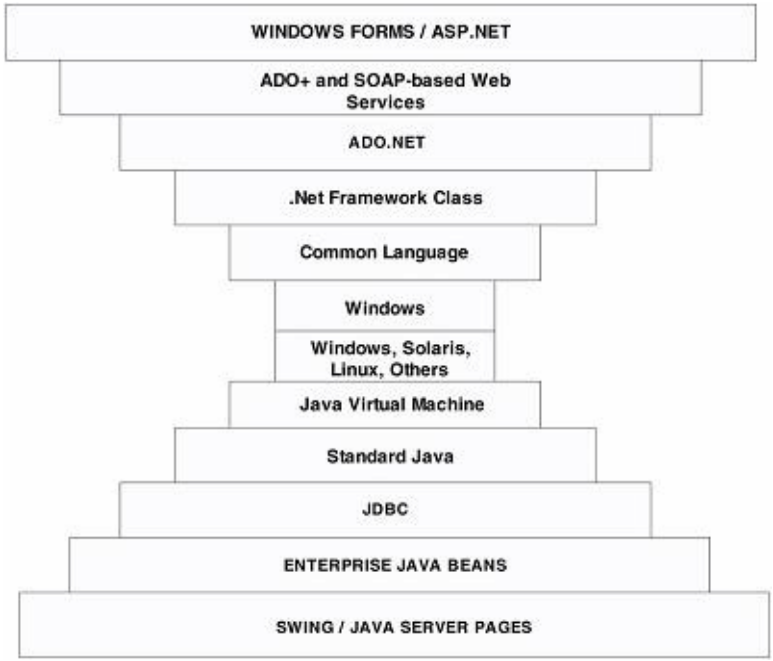
When J2EE or .NET applications are running, the operating system on which they run provides various parameters that can be set to specific values for optimal performance. Such parameters are monitored and measured by various counters. Knowing the counters that help in tuning the operating system from a performance point of view is of paramount importance for all test analysts.

The chapters that follow discuss the important counters related to the Windows and UNIX systems as most of the applications run on these two operating systems.

In UNIX, the major resource types that need to be monitored and tuned are the CPU, memory, disk space, communication lines, I/O time, network time, and application programs. The UNIX operating system maintains several counters that keep track of the system resources and their utilization. Some of these counters are the CPU utilization, buffer usage, disk I/O activity, tape I/O activity, terminal activity, system call activity, context switching activity, file access utilization, queue activity, interprocess communication (IPC), paging activity, free memory and swap space, kernel memory allocation (KMA), and so on. For details, see Chapter 4, "Monitoring UNIX."

Windows is a **self-tuning** operating system. This means that in most cases, Windows automatically adapts to perform optimally depending on the environment in which it is running, assuming the hardware is properly configured. For instance, when Windows is deployed as a Web server, other services that are also present but are not used are put into a state where they occupy very few system resources such as CPU and memory. However, like many other operating systems, performance depends on many outside factors such as hardware, device drivers, applications, workload, network, and so on. For details, see Chapter 3, "Windows Monitoring."

Both J2EE and .NET require the application architecture to be defined in advance, before the development of the application. These technologies support their own frameworks for defining the architecture. However, there are certain architectural similarities between these technologies to define the system. These similarities help us to define common guidelines for monitoring performance counters and tuning applications. J2EE and Microsoft's .NET technology share a broad common foundation of standards, and they both have adopted the multi-tiered architecture approach that typically implements applications in different logical layers, which separate presentation from internal structure (business logic and data management):



- Both J2EE and .NET architecture models use the object oriented (OO) approach for mainstream enterprise computing, with powerful OO frameworks (class libraries) for services such as enterprise components management, object persistence, transactions, Web services, asynchronous communication, loosely coupled event services, messaging, and more.
- The use of virtual machine (VM) architecture is common to J2EE and .NET. Application development tools produce intermediate level code instead of platform-specific binary code. This means that the VM interprets the code in real time or performs Just-In-Time (JIT) compilation.
- J2EE and .NET share a broad common foundation that implements the multi-tiered approach.

During QA cycles, load testing typically follows integrated functional and regression testing. You should load test a complete application, including all interfaces with external systems, before releasing the software.

Objectives include estimating scalability and capacity under a load that realistically represents expected live use, along with gaining visibility into the internal performance behavior of the application and gathering actionable data on bottlenecks. This should include a transaction breakdown of latencies for each J2EE/.NET tier and method, along with additional specific root cause diagnostic information.

6

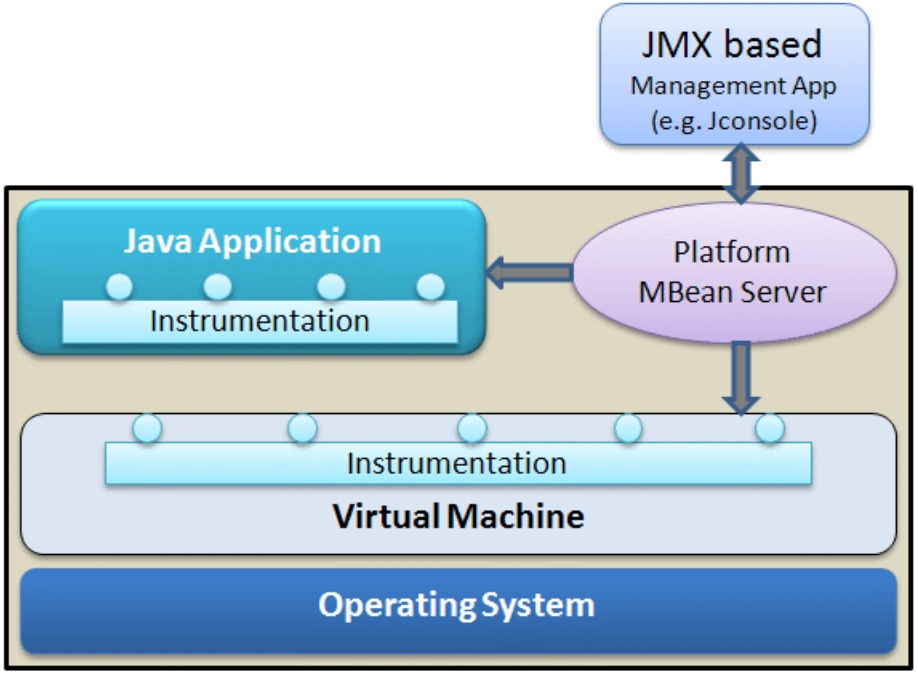
Java Platform Monitoring

This chapter describes best practices for Java platform monitoring.

This chapter includes:

Introduction

The Java 2 platform provides comprehensive monitoring and management support. It not only defines the management interfaces for the Java Virtual Machine (JVM), but also provides out-of-the-box remote monitoring and management on the Java platform and of applications that run on it.



In addition, JDK 5.0 includes the Java Monitoring and Management Console tool **(JConsole**). JDK 5.0 uses the extensive instrumentation of the JVM to provide information on performance and resource consumption of applications running on the Java platform using Java Management Extension (JMX) technology. JMX provides a standard way to instrument the Java runtime environment and applications. The instrumentation is accessible through the JMX managed bean (MBean) interfaces, which are registered in the platform MBean server. Applications can also create their own MBeans and register them in the platform MBean server, which can serve as a single point for remote access. A JMX-compliant client, such as JConsole, can connect to the platform MBean server and manage the application (as well as the Java platform) using JMX technology. Each platform MBean has a rich set of attributes and operations such as memory usage, thread CPU usage, garbage collection statistics, and so on.

HP SiteScope provides built-in support for JMX, rendering usage of JConsole unnecessary and giving a consolidated view of both operating system counters and Java-specific application measurements. All counters available through JConsole are also accessible via HP SiteScope.

Most Important Java Counters

	Counter	Description
Common	Uptime	Indicates how long the JVM has been running.
	Total compile time	Indicates the amount of time spent in just-in-time (JIT) compilation.
	Process CPU time	Indicates the total amount of CPU time consumed by the JVM.
Memory	Current heap size	Indicates the number of kilobytes currently occupied by the heap.
	Maximum heap size	Indicates the maximum number of kilobytes occupied by the heap.
	Committed memory	Indicates the total amount of memory allocated for use by the heap.
	GC time	Indicates the cumulative time spent on garbage collection and the total number of invocations.
Threads	Live threads	Indicates the current number of live daemon threads plus non-daemon threads.
	Peak threads	Indicates the highest number of live threads since JVM started.
	Daemon threads	Indicates the current number of live daemon threads.
	Total started threads	Indicates the total number of threads started since JVM started (including daemon, non-daemon, and terminated).
Classes	Current classes loaded	Indicates the number of classes currently loaded into memory.
	Total classes loaded	Total number of classes loaded into memory since the JVM started, included those subsequently unloaded.
	Total classes unloaded	Number of classes unloaded from memory since the JVM started.

Common Counters

This section describes the counters that show common information pertaining to JVM running on the machine.

Uptime

Official Name	Uptime
Counter Type	Elapsed time
Description	Amount of time passed since JVM started on the machine
Usage Notes	Shows overall status of Java
Performance	Important indicator of overall health
Operations	The longer JVM is running, the more threads may remain open if garbage collection is running rarely
Threshold	N/A

Total compile time

Official Name	Total compile time
Counter Type	Elapsed time
Description	The amount of time spent in just-in-time (JIT) compilation. The JVM implementation determines when JIT compilation occurs.
Usage Notes	Since JVM interprets Java into bytecode, it needs to compile objects upon load. This counter shows how much time has been spent overall on such compilations since JVM started running. Sun's Hotspot VM uses adaptive compilation, in which the VM launches an application using a standard interpreter, but then analyzes the code as it runs to detect performance bottlenecks, or "hot spots".
Performance	Secondary indicator to determine if a large number of new objects poses a potential bottleneck
Operations	This counter can pinpoint whether a system is properly deployed and initiated
Threshold	N/A

Process CPU time

Official Name	Process CPU time
Counter Type	Elapsed time
Description	The total amount of CPU time consumed by the JVM
Usage Notes	One of the main indicators to see how JVM affects overall operating system behavior
Performance	May be used to calculate the percentage of time the processor spends on Java and all other processes. A dramatic increase in this counter indicates potential problems.
Operations	This counter may pinpoint necessary changes to scale up JVM
Threshold	Depends on the processor

Memory Counters

This section describes the counters that usually appear on the Memory tab of JConsole. They display data about memory consumption, memory pools, and garbage collection statistics.

The memory pools available depend on the JVM being used. The following list shows the pools for the HotSpot virtual machine which comes with standard installation of Sun Java.

- **Eden Space (heap) Pool**. Memory is initially allocated for most objects from this pool.
- **Survivor Space (heap) Pool.** Contains objects that have survived garbage collection of the Eden Space pool.
- **Tenured Generation (heap) Pool.** Contains objects that have existed for some time in the Survivor Space pool.

- **Permanent Generation (non-heap) Pool.** Holds all the reflective data of the virtual machine itself, such as class and method objects. With JVMs that use class data sharing, this pool is divided into read-only and read-write areas.
- **Code Cache (non-heap) Pool.** The HotSpot JVM also includes a "code cache" that contains memory used for compilation and storage of native code.

Each memory pool may have two kinds of memory thresholds for low memory detection support: a **usage threshold** and a **collection usage threshold**. Either one of these thresholds might not be supported by a particular memory pool.

- **Usage threshold**. A manageable attribute of a memory pool. It enables the monitoring of memory use with low overhead. Setting the threshold to a positive value enables usage threshold checking for a memory pool. Setting the usage threshold to **zero** disables usage threshold checking. The default value is supplied by the JVM. A JVM performs usage threshold checking on a memory pool at the most appropriate time, typically during garbage collection and sometimes at allocation time. If the JVM detects that the current memory usage exceeds the usage threshold, it will set the **UsageThresholdExceeded** attribute to **true**.
- **Collection usage threshold.** A manageable attribute of some garbage-collected memory pools. After a JVM has performed garbage collection on a memory pool, some memory in the pool is still be occupied by reachable objects. The collection usage threshold allows you to set a value to check against the memory usage only after garbage collection. If the JVM detects that the memory usage exceeded the collection usage threshold, it sets the **CollectionUsageThresholdExceeded** attribute to **true**.

The JVM manages two kinds of memory which are both created when the JVM starts:

- **Heap memory.** The runtime data area from which the JVM allocates memory for all class instances and arrays. The heap may be of a fixed or variable size. The garbage collector is an automatic memory management system that reclaims heap memory for objects.
- **Non-heap memory.** Includes a method area shared among all threads and memory required for the internal processing or optimization for the JVM. It stores per-class structures such as a runtime constant pool, field and method data, and the code for methods and constructors. The method area is logically part of the heap but, depending on implementation, a JVM may not collect garbage or compact it. Like the heap, the method area may be of fixed or variable size. The memory for the method area does not need to be contiguous.

  In addition to the method area, a JVM implementation may require memory for internal processing or optimization which also belongs to non-heap memory. For example, the JIT compiler requires memory for storing the native machine code translated from the JVM code for high performance.

Current heap size

Official Name	Current heap size
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The amount of memory currently used
Usage Notes	Memory used includes the memory occupied by all objects including both reachable and unreachable objects
Performance	Important indicator to determine whether the memory is a potential bottleneck
Operations	If this parameter increases over time, it may indicate a need for reconfiguration
Threshold	See explanation above

Maximum heap size

Official Name	Maximum heap size
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The maximum amount of memory that can be used for memory management. Its value may change or be undefined
Usage Notes	A memory allocation may fail if the JVM attempts to increase the used memory to be greater than committed memory, even if the amount used is less than or equal to max (for example, when the system is low on virtual memory)
Performance	Shows upper memory limit - warning if close to physical memory boundaries
Operations	If not implicitly defined, may cause improper memory allocation
Threshold	See explanation above

Committed memory

Official Name	Committed memory
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The total amount of memory allocated for use by the heap
Usage Notes	The amount of memory guaranteed to be available for use by the JVM. The amount of committed memory may change over time. The JVM may release memory to the system and the committed memory could be less than the amount of memory initially allocated at startup
Performance	N/A
Operations	N/A
Threshold	Committed will always be greater than or equal to used

GC time

Official Name	GC (Garbage Collection) time
Counter Type	Elapsed time
Description	The cumulative time spent on garbage collection and the total number of invocations. It may have multiple rows, each representing one garbage collector algorithm used in the JVM.
Usage Notes	Garbage collection (GC) is how the JVM frees memory occupied by objects that are no longer referenced. It is common to think of objects that have active references as being **alive** and of non-referenced (unreachable) objects as **dead**. Garbage collection is the process of releasing memory used by the dead objects.
Performance	The algorithms and parameters used by GC can have dramatic effects on performance. Sun's HotSpot VM garbage collector uses **generational** garbage collection. Generational GC utilizes the fact that, in practice, most programs create: ➤ many objects that have short lives (for example, iterators and local variables) ➤ some objects that have very long lives (for example, high-level persistent objects) So, generational GC divides memory into several generations, and assigns each a memory pool. When a generation uses up its allotted memory, the VM performs a partial garbage collection (also called a **minor** collection) on that memory pool to reclaim memory used by dead objects. This partial GC is usually much faster than a full GC.
Operations	If GC has become a bottleneck, you may want to customize the generation sizes. Check the verbose GC output, and then explore the sensitivity of your individual performance metric to the GC parameters.
Threshold	Committed will always be greater than or equal to used.

Note: One of the most bothersome experiences for users with less than ideal memory configurations is GC pauses. There are a number of settings that affect the way the JVM allocates memory and the behavior of GC. The main purpose of monitoring GC—and hence tuning—is to reduce the frequency of major GC events without increasing their accumulating duration.

Thread Counters

A **thread** relates to a thread of execution in a program. The JVM allows an application to have multiple threads of execution running concurrently. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. Each thread may or may not also be marked as a daemon. When code running in some thread creates a new Thread object, the new thread has its priority initially set equal to the priority of the creating thread, and is a daemon thread if, and only if, the creating thread is a daemon.

When a JVM starts up, there is usually a single non-daemon thread (which typically calls the method named **main** of some designated class). The JVM continues to execute threads until either of the following has occurred:

- The exit method of class Runtime has been called and the security manager has permitted the exit operation to take place.
- All threads that are not daemon threads have died, either by returning from the call to the run method or by throwing an exception that propagates beyond the run method

Live threads

Official Name	Live threads
Counter Type	Instantaneous (sampled once during each measurement period)
Description	Shows the current number of live daemon threads plus non-daemon threads
Usage Notes	N/A
Performance	Too many threads may cause slow garbage collection operation
Operations	N/A
Threshold	Watch when this counter approaches Peak threads value

Peak threads

Official Name	Peak threads
Counter Type	Cumulative
Description	Highest number of live threads since JVM started
Usage Notes	May be helpful in recognizing trending patterns of JVM behavior
Performance	N/A
Operations	N/A
Threshold	N/A

Daemon threads

Official Name	Daemon threads
Counter Type	Instantaneous (sampled once during each measurement period)
Description	Current number of live daemon threads
Usage Notes	N/A
Performance	Too many threads may cause slow garbage collection operation
Operations	N/A
Threshold	Watch when this counter approaches Peak threads value

Total started threads

Official Name	Total started threads
Counter Type	Cumulative
Description	Total number of threads started since JVM started (including daemon, non-daemon, and terminated)
Usage Notes	May be helpful in recognizing trending patterns of JVM behavior
Performance	N/A
Operations	N/A
Threshold	N/A

Note: It is usually enough to monitor only one pair of thread counters, such as Total Started Threads and Live Threads, as the other ones can be derived from them.

Tip: To check if your application has run into a deadlock (for example, your application seems to be hanging), you can invoke the **findMonitorDeadlockedThreads** operation from JConsole's MBeans tab.

Class Counters

This section describes the most important class counters.

Current classes loaded

Official Name	Current classes loaded
Counter Type	Instantaneous (sampled once during each measurement period)
Description	The number of classes currently loaded into memory
Usage Notes	N/A
Performance	N/A
Operations	N/A
Threshold	N/A

Total classes loaded

Official Name	Total classes loaded
Counter Type	Cumulative
Description	Total number of classes loaded into memory since the JVM started, including those subsequently unloaded
Usage Notes	May be helpful in recognizing trending patterns of JVM behavior
Performance	N/A
Operations	N/A
Threshold	N/A

Total classes unloaded

Official Name	Total classes unloaded
Counter Type	Cumulative
Description	Shows number of classes unloaded from memory since the JVM started
Usage Notes	May be helpful in recognizing trending patterns of JVM behavior
Performance	Non-zero value of this counter over long period of time may point to problems with garbage collection mechanism
Operations	N/A
Threshold	N/A

7

.NET Platform Monitoring

This chapter describes the best practices for .NET platform monitoring.

This chapter includes:

Introduction

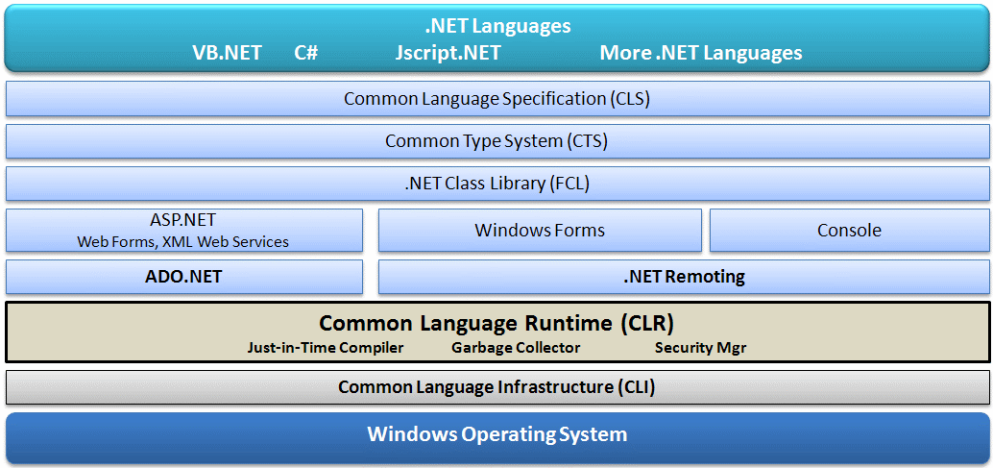
Most of the applications developed with Microsoft technology use the .NET framework. This framework provides a good platform for both the development and running of applications. It also provides counters to measure and monitor performance of the applications.

The .NET Framework has two main components:

- The common language runtime
- The .NET Framework class library

The common language runtime (CLR) is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library, the other main component of the .NET Framework, is a comprehensive, object-oriented collection of reusable types.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called **just-in-time (JIT) compiling** enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to further increase performance.



Performance counters are organized and grouped into performance counter categories. In general, just as the Windows operating system provides many predefined performance counters that can be retrieved programmatically or displayed using the Performance Monitor, in .NET the CLR exposes its own set of performance counters. They are organized into nine important categories to help the tester monitor and tune the application's performance. They are:

- **Exceptions.** Provides information about the exceptions thrown by the application.
- **Memory.** Provides information about the garbage collector.
- **Locks and Threads.** Provide information about managed locks and threads used by the application.
- **Interop.** Provides information about the application's interaction with COM components, COM+ services, and type libraries.

- **JIT.** Provides information about code that has been compiled by the just-in-time (JIT) compiler.
- **Loading.** Provides information about assemblies, classes, and AppDomains that have been loaded.
- **Networking.** Provides information about the data sent and received over the network by the application.
- **Remoting.** Provides information about remote objects used by the application.
- **Security.** Gives a description about the security checks the CLR performs on the application.

Most Important .NET Counters

When monitoring .NET applications, it is recommended to start monitoring from operating system counters that measure the utilization of the processors, the memory, the network, and the I/O devices (see Windows chapter). Then you can add .NET performance counters that cover every aspect of the CLR operations ranging from exception processing to security checking.

	Counter	Description
Exception	# of Excep Thrown/sec	Indicates the number of managed code exceptions thrown per second
	Throw to Catch Depth/Sec	Indicates the number of stack frames

	Counter	Description
Memory	Large Object Heap Size	Indicates the current size of the Large Object Heap in bytes
	# Bytes in all Heaps	Indicates the current memory allocated in bytes on the GC heaps.
	# of Pinned Objects	Indicates the number of pinned objects encountered in the last garbage collection (GC).
	% Time in GC	Indicates the percentage of elapsed time that was spent in performing a garbage collection (GC) since the last GC cycle.
Threads	# of Current Logical Threads	Indicates the number of current .NET thread objects in the application.
	# of Current Physical Threads	Indicates the number of native OS threads created and owned by the CLR.
	# of Current Recognized Threads	Indicates the number of threads currently recognized by the CLR.
	# of Total Recognized Threads	Indicates the total number of threads recognized by the CLR since the start.
	Contention Rate/Sec	Indicates the rate at which threads in the runtime attempt to acquire a managed lock unsuccessfully
Loading	Current Assemblies	Indicates the number of assemblies that are loaded in the process.
	Rate of Assemblies	Indicates the rate at which assemblies are loaded into the memory per second.
	Bytes in Loader Heap	Indicates the number of bytes committed by the class loader.
Security	Total Runtime Checks	Indicates the percentage of elapsed time spent in performing runtime Code Access Security.
	Stack Walk Depth	Indicates the depth of the stack during that last runtime Code Access Security check.

Exception Counters

This section describes the counters that provide information pertaining to exceptions thrown by the .NET application.

of Excep Thrown/sec

Official Name	.NET CLR Exception/# of Excep Thrown/sec (_Global_)
Counter Type	Instantaneous (sampled once during each measurement period).
Description	This counter displays the number of exceptions thrown per second. These include both .NET exceptions and unmanaged exceptions.
Usage Notes	This counter includes both handled and unhandled exceptions. Exceptions should only occur in rare situations and not in the normal control flow of the program.
Performance	Indicator of potential performance problems due to large rate of exceptions thrown.
Operations	This has to be 0 under normal circumstances.
Threshold	Error if rate is larger than 100.

Throw to Catch Depth/Sec

Official Name	.NET CLR Exception/Throw to Catch Depth/Sec.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Displays the number of stack frames traversed from the frame that threw the .NET exception to the frame that handled the exception per second.
Usage Notes	Resets to 0 when an exception handler is entered; so nested exceptions would show the handler to handler stack depth.
Performance	Secondary indicator to determine code shortcomings that may pose a potential bottleneck.
Operations	N/A
Threshold	N/A

Memory Counters

This section describes the counters that belong to memory management of .NET CLR. They provide data about memory consumption, memory pools, and garbage collection statistics.

The common language runtime's garbage collector (GC) manages the allocation and release of memory for an application. This automatic memory management can eliminate common problems, such as forgetting to free an object and causing a memory leak, or attempting to access memory for an object that has already been freed.

When a .NET application is initialized, the runtime reserves a contiguous region of address space for the process. This reserved address space is called the **managed heap**. When an application is created the first object, memory, is allocated at the base address of the managed heap. When the application creates the next object, the garbage collector allocates memory for it in the address space immediately following the first object. As long as address space is available, the garbage collector continues to allocate space for new objects in this manner. Allocating memory from the managed heap is faster than **unmanaged** memory allocation. Unmanaged resources require explicit cleanup as GC is not always able to trace the execution stack.

To optimize the performance of the garbage collector, the managed heap is divided into three generations: **Gen 0**, **Gen 1**, and **Gen 2**. The runtime's garbage collector stores new objects in **Gen 0**. Objects created early in the application's lifetime that survive collections are promoted and stored in **Gen 1** and **Gen 2**. This scheme allows the garbage collector to release the memory in a specific generation faster, rather than release the memory for the entire managed heap each time it performs a collection.

Large Object Heap Size

Official Name	.NET CLR Memory\Large Object Heap Size.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The amount of memory in bytes currently used by Large Object Heap.
Usage Notes	Objects greater than 20KBytes are treated as large objects by the garbage collector and are directly allocated in a special heap, which is measured by High CPU utilization.
Performance	Important indicator to determine code deficiency as releasing the entire heap takes more time than when generational algorithm works properly. Usually called Fragmented Large Object heap bottleneck.
Operations	Large Objects are not promoted through the generations.
Threshold	N/A

Bytes in all Heaps

Official Name	.NET CLR Memory\# Bytes in all Heaps
Counter Type	Instantaneous (sampled once during each measurement period)
Description	Shows the current memory allocated in bytes on the GC heaps.
Usage Notes	This counter is the sum of the **Gen 0 Heap Size, Gen 1 Heap Size, Gen 2 Heap Size,** and the **Large Object Heap Size** counters. This counter indicates the current memory allocated in bytes on the garbage collection heaps.
Performance	While using large data sets in memory, excess cache entries, using reg-ex and string parsing, excess view-state or excessive session objects contribute to the heavy memory requirement.
Operations	You usually start monitoring from this selecting counter. An increase in **Private Bytes** while the **# of Bytes in all heaps** counter remains the same indicates unmanaged memory consumption. An increase in both counters indicates managed memory consumption.
Threshold	Should be less than the Process\Private Bytes counter

of Pinned Objects

Official Name	.NET CLR Memory\# of Pinned Objects.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Displays the number of pinned objects encountered in the last GC.
Usage Notes	A pinned object is one that the garbage collector cannot move in memory. This counter tracks the pinned objects only in the heaps that were garbage collected; for example, a **Gen 0** garbage collection would cause enumeration of pinned objects in the **Gen 0** heap only.
Performance	N/A
Operations	N/A
Threshold	N/A

% Time in GC

Official Name	.NET CLR Memory\% Time in GC.
Counter Type	Elapsed time.
Description	The percentage of elapsed time that was spent performing a garbage collection (GC) since the last GC cycle.
Usage Notes	An indicator of the work done by the garbage collector on behalf of the application to collect and compact memory.
Performance	Allocating large strings to cache, heavy string operations, and so on, leave a lot of memory spaces the GC has to clean up.
Operations	Updated only at the end of every GC, and the counter value reflects the last observed value; it is not an average. If there are any spikes in this counter, then those are accepted.
Threshold	Should be in the range of 5-10%.

Thread Counters

A **thread** relates to a thread of execution in a program. A .NET **logical thread** object is created either implicitly by issuing a new **System.Threading.Thread** command inside the code or explicitly when an unmanaged thread enters the managed environment. There are also physical threads, created and owned by the CLR which are essentially native OS acting as underlying threads for .NET thread objects. The .NET application may make usage of recognized threads which are not created by the CLR - they are created outside the CLR but have since run inside the CLR at least once.

of Current Logical Threads

Official Name	.NET CLR LocksAndThreads\# of Current Logical Threads.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Displays the number of current managed thread objects in the application.
Usage Notes	This counter maintains the count of both running and stopped threads.
Performance	Too many threads may cause slow garbage collection operation.
Operations	N/A
Threshold	N/A

of Current Physical Threads

Official Name	.NET CLR LocksAndThreads\# of Current Physical Threads.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Represents the number of native operating system threads created and owned by the common language runtime to act as underlying threads for managed thread objects.
Usage Notes	This is subset of the threads in the OS process.
Performance	N/A
Operations	This counter does not include the threads used by the CLR in its internal operations.
Threshold	N/A

of Current Recognized Threads

Official Name	.NET CLR LocksAndThreads\# of Current Recognized Threads.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Shows the number of threads currently recognized by the CLR.
Usage Notes	Only unique threads are tracked.
Performance	N/A
Operations	Threads with the same thread ID reentering the CLR or recreated after thread exit are not counted twice.
Threshold	N/A

of Total Recognized Threads

Official Name	.NET CLR LocksAndThreads\# of Total Recognized Threads.
Counter Type	Cumulative.
Description	Total number of threads recognized by the CLR since the start of this application.
Usage Notes	Only unique threads are tracked.
Performance	N/A
Operations	Threads with the same thread ID reentering the CLR or recreated after thread exit are not counted twice.
Threshold	N/A

Contention Rate/Sec

Official Name	.NET CLR LocksAndThreads\Contention Rate/Sec.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Rate at which threads in the runtime unsuccessfully attempt to acquire a managed lock.
Usage Notes	An increase in the contention rate or a significant increase in the total number of contentions is a strong indication that an application is encountering thread contention. To resolve this issue, one has to identify code that accesses shared resources or uses synchronization mechanisms.
Performance	Along with Total Number of Contentions, may point to thread bottleneck.
Operations	N/A
Threshold	N/A

Loading Counters

This section describes the most important loading counters.

Current Assemblies

Official Name	.NET CLR Loading/Current Assemblies (_Global_).
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Indicates and records the number of assemblies that are loaded in the process.
Usage Notes	This value is accumulated across all application domains in the currently running application.
Performance	N/A
Operations	If the assembly is loaded as domain-neutral from multiple application domains, this counter is incremented only once.
Threshold	N/A

Rate of Assemblies

Official Name	.NET CLR Loading/Rate of Assemblies.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Rate at which assemblies are loaded into the memory per second.
Usage Notes	This value is accumulated across all application domains in the currently running application.
Performance	N/A
Operations	If the assembly is loaded as domain-neutral from multiple application domains, this counter is incremented only once.
Threshold	N/A

Bytes in Loader Heap

Official Name	.NET CLR Loading/Bytes in Loader Heap.
Counter Type	Cumulative.
Description	Indicates the number of bytes committed by the class loader across all application domains.
Usage Notes	Committed memory is the physical space reserved in the disk paging file.
Performance	This counter has to be in a steady state, or else large fluctuations in this counter would indicate that there are too many assemblies loaded per application domain.
Operations	N/A
Threshold	N/A

Security Counters

This section describes the most important security counters.

Total Runtime Checks

Official Name	.NET CLR Security/Total Runtime Checks.
Counter Type	Cumulative.
Description	Displays the total number of runtime Code Access Security (CAS) checks since the application started.
Usage Notes	CAS allows code to be trusted to varying degrees and enforces these varying levels of trust depending on code identity. Runtime code access security checks are performed when a caller demands a particular permission. The runtime check is made on every call by the caller and examines the current thread stack of the caller.
Performance	When used with the **Stack Walk Depth** counter, this counter indicates the performance penalty that occurs for security checks.
Operations	This counter is updated at the end of a runtime security check.
Threshold	N/A

Stack Walk Depth

Official Name	.NET CLR Security/Stack Walk Depth.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	This counter displays the depth of the stack during that last runtime Code Access Security check.
Usage Notes	The Runtime Code Access Security check is performed by crawling the stack.
Performance	N/A
Operations	When used with the **Total Runtime Checks** counter, this counter indicates the performance penalty that occurs for security checks.
Threshold	N/A

Part IV

Web Server Monitoring

8

Apache Monitoring

This chapter describes best practices for Apache monitoring.

This chapter includes:

Overview

The Apache HTTP server is an open source, configurable and extensible, multi-platform Web server. It was initially developed in 1995 using NCSA httpd (HTTP daemon) as a base. In time, the Apache HTTP server became one of the most commonly used Web servers for commercial Web sites and Web-based applications.

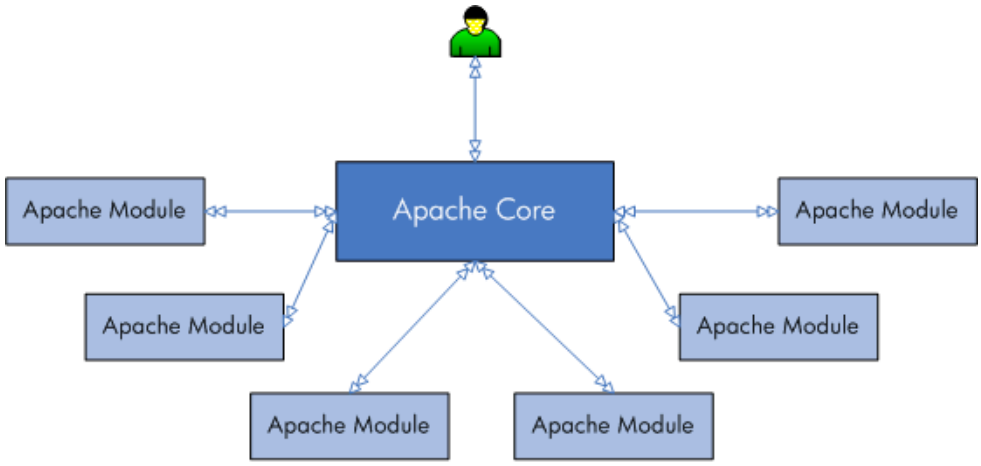
As one of the dominant Web servers it is important to understand Apache's high level architecture, counters for monitoring, tuning aspects ,and other performance related best practices. This chapter summarizes these aspects as well as making you familiar with LoadRunner & Performance Center techniques for monitoring the Apache Web server.

Architecture

The function of a Web server is to service requests made through the HTTP protocol. Typically the server receives a request asking for a specific resource and returns the resource as a response back to the client. Apache fulfills this purpose by separating the request handling responsibility to the Apache core and Apache modules:

- The core is responsible for defining and following the steps in servicing a request.
- The modules actually implement the different phases involved in handing a request.

This architecture makes Apache a great platform for third parties to override or extend functionality as well as allow Administrators to tune the server for best memory management by turning off unused modules.



High Level Conceptual Architecture

Apache 2.0 architecture and capabilities are superior to those of Apache 1.3, even though both series are considered to be production quality versions. The following describe performance related characteristics in Apache 2.0 architecture:

- **Multi Processing Modules.** Apache 2.0 supports Multi Processing Modules (MPMs) as opposed to Apache 1.3 which is process-based that forks several children at startup. MPMs mean that Apache can be configured to be a pure process-based server, a purely threaded server or a mixture of those models. Threads are contained inside processes and run simultaneously and in most cases threaded servers scale better than process based servers.
- **Module and filter.** As mentioned above, Apache maintains modular architecture. Apache 2.0 adds an additional extension mechanism: filters. Filters allow modules to modify the content generated by other modules. They can encrypt, scan for viruses or compress not only static files but dynamically generated content.

➤ **Apache Portable Runtime.** Apache 2.0 runs equally well on Windows and UNIX platforms thanks to the Apache Portable Runtime (APR) library. It abstracts the differences among operating systems, such as file or network access APIs. This abstraction layer also provides for platform-specific tuning and optimization. The APR uses the concept of memory pools, which significantly simplifies the memory-management code and reduces the possibility of memory leaks.

In compliancy with Apache architecture, counters exposed by Apache for understanding and monitoring server status are available from the Apache **mod_status** module. The status module provides information on server activity and performance. It exposes the server statistics either in the HTML page in an easily readable form (i.e. http://your.server.name/server-status) or in a simple machine-readable list (i.e. http://your.server.name/server-status?auto) oriented for automating the monitoring process. Both modes can be configured to automatically refresh the status by adding refresh parameter in the URL query string (for example, http://your.server.name/server-status?auto&refresh=30 will automatically refresh the machine-readable status every 30 seconds).

The mod_status module can be configured to provide extended status. By default it is disabled.

Note: The Apache monitor connects to the Web server in order to gather statistics, and registers one hit for each sampling. The Apache graph, therefore, always displays at least one hit per second, even if no clients are connected to the Apache server.

Most Important Apache Counters

The HP LoadRunner/Performance Center Apache monitor is built to track the counters exposed in the machine-readable page (server-status?auto). HP SiteScope supports both modes. The most important counters are available in the machine-readable page.

Counter	Description
CPULoad	The current percentage of CPU consumed by the Apache server
ReqPerSec	The number of requests per second (a.k.a. hits per second)
BytesPerSec	The number of bytes transferred per second
BytesPerReq	The number of bytes transferred per request
BusyWorkers	Number of active threads serving requests
IdleWorkers	Number of inactive/idle threads

Tip: All these counters are available in the server-status?auto page. You can easily create a VuGen script to parse these counters data on your own and send it to LoadRunner/Performance Center online using lr_user_data_point.

Optimization and Tuning

When performance issues are encountered, tuning and optimization are required to alleviate the issues. It is recommended to act proactively and prevent these issues from occurring in the first place. This section lists a few possible tuning parameters, optimization practices, and benchmarking methods oriented for the Apache Web server. Before applying any of these, you should first validate the relevancy of the configuration to your specific case by understanding the parameter and the workload generated against your server.

- HostnameLookups directive should be off (off by default). When turned on DNS lookups will consume a lot of time and slow the server.

- KeepAlive directive should be on (on by default). KeepAlives provide longer HTTP sessions which allow multiple requests to be sent over the same TCP connection. This has proved to be extremely significant for speeding up response time.

- KeepAliveTimeout directive represents the number of seconds Apache will wait for a subsequent request before closing a connection. The default configuration is 15 seconds. The higher the timeout, the more server threads will be kept occupied waiting on connections with idle clients.

- Avoid using .htaccess files. The use of .htaccess files can be disabled completely by setting the AllowOverride directive to **none**. When AllowOverride is set to allow the use of .htaccess files, Apache will look in every directory for .htaccess files. Permitting .htaccess files causes a performance hit, whether or not you actually even use them. Also, the .htaccess file is loaded every time a document is requested.

- It is recommended to unload unused modules in order to optimize memory utilization.

- MaxKeepAliveRequests directive. A Web server should never have to swap, as swapping increases the latency of each request beyond a point that users consider "fast enough". This causes users to hit stop and reload, further increasing the load. You can, and should, control the MaxClients setting to prevent your server from spawning so many children that it starts swapping. The MaxKeepAliveRequests directive specifies the maximum number of child processes that will be created to serve requests, and limits the number of simultaneous requests that will be served. Any connection attempts that are over the MaxClients limit will normally be queued, up to a number based on the ListenBacklog directive. You should set this to the maximum number of clients that your environment can manage without experiencing throughput degradation or a prohibitive increase of the response time.

9

IIS Monitoring

This chapter describes best practices for IIS monitoring.

This chapter includes:

Overview

Microsoft Internet Information Services (IIS) is the world's second most popular Web server after the Apache HTTP Server. IIS is available within all Windows operating system editions in different flavors. It is constantly evolving and maturing. IIS 6.0 made a major step forward in enabling using IIS not only as a Web server but also as an application server. IIS 7.0, the latest IIS release, continues this path by adding important capabilities that contribute also to performance and reliability.

IIS includes the following servers: FTP/FTPS, SMTP, NNTP, and HTTP/HTTPS. This chapter focuses on the HTTP/HTTPS server. It covers IIS architecture, performance monitoring, and some tuning guidelines. The focus in this chapter is mainly on IIS 6.0, although IIS 7.0 is mentioned where necessary.

Architecture

IIS runs a server in one of two distinct request processing models, called application isolation modes. Application isolation is the separation of applications by process boundaries that prevents one application or Web site from affecting another and reduces the time spent restarting services to correct problems related to applications.

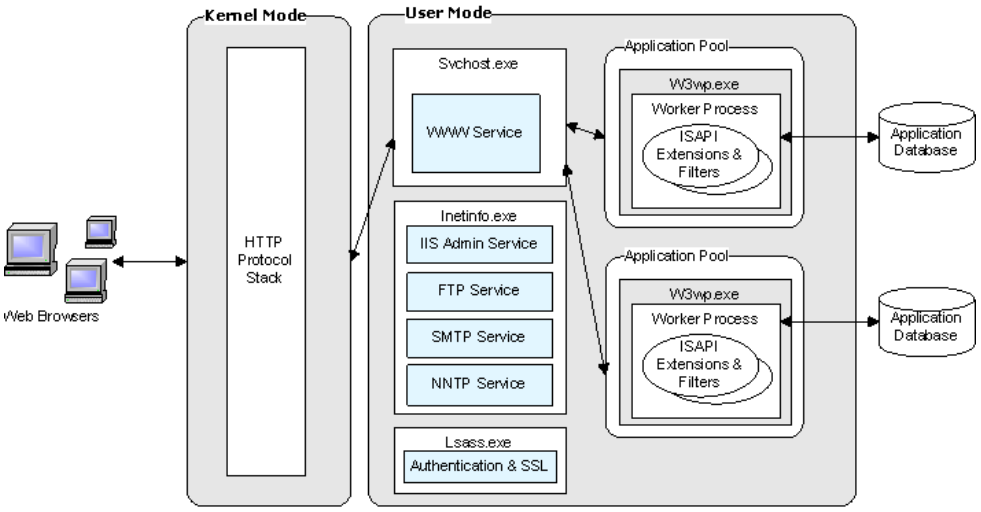
IIS 6.0 supports two application isolation modes. Each mode has a different configuration:

- **Worker Process Isolation mode.** Supports grouping Web applications into application pools thus enabling each application to function within a self-contained worker process. A worker process is user-mode code whose role is to process requests, such as returning a static page or invoking an Internet Server API (ISAPI) extension or filter. This mode delivers all the benefits of IIS 6.0 architecture, including multiple application pools, health monitoring and recycling, increased security and performance, improved scalability, and processor affinity.
- **IIS 5.0 Isolation mode.** Provides compatibility for applications that were designed to run in earlier versions of IIS. When IIS 6.0 is running in this mode, request processing is almost identical to the request processing in IIS 5.0. Unless your application does not function in worker process isolation mode, it is not recommended to use this mode.

Both modes rely on the HTTP protocol stack (HTTP.sys) to receive HTTP requests and return responses. HTTP.sys listens for HTTP requests, queues them and return responses after the requests where processed.

HTTP.sys resides in Kernel Mode where operating system code, such as device drivers, runs. This ensures that the operating system manages HTTP requests in high priority. The actual processing of the request is done in User Mode by the relevant Worker process.

The following diagram illustrates the Worker Process Isolation mode.



The application pool can host multiple Worker processes, thus provides load balancing and failover capabilities. This contributes to the performance, reliability, and scalability of the application. An application pool that contains more than one Worker processes is called a Web garden.

As mentioned earlier, IIS provides four internet services: the World Wide Web Publishing Service (WWW service) for hosting Internet and intranet content; the File Transfer Protocol (FTP) service for hosting sites where users can upload and download files; the Network News Transfer Protocol (NNTP) service for hosting discussion groups; and the Simple Mail Transfer Protocol (SMTP) service for sending and receiving email messages. It is recommended to disable/uninstall unused services in order to reduce IIS footprint.

IIS 7.0 introduces a few architectural enhancements:

- A new service, Windows Process Activation Service (WAS), was added. Enables sites to use protocols other than HTTP/HTTPS.
- Integration of request processing pipelines from IIS and ASP.NET. This capability is related to the application pool modes supported in IIS 7.0.
  - IIS 5.0 isolation mode is no longer supported
  - IIS 6.0 Worker Process Isolation mode continues to be supported
  - A new mode, Integrated Application Pool mode, was added in order to allow integrated request processing of IIS and ASP.NET
- Web Server engine can be customized by adding or removing modules

Monitoring

IIS performance counters are exposed through the Microsoft Windows performance data helper library (pdh.dll) which is the general monitoring platform of Windows. This means that each IIS performance counter is numeric and is uniquely identified by its path, usually in the following syntax:

```
\\Computer_name\Object(Parent/Instance#Index)\Counter
```

The **Computer_name** portion of the path is optional.

Both SiteScope and LoadRunner use the Windows pdh interface for monitoring IIS and ASP/ASP.NET related counters. To invoke the pdh interface from a remote machine, Windows requires authentication using a user that has appropriate permissions.

As a best practice, it is recommended to gain good understanding of your application architecture and deployment. This information is useful while performing different performance engineering practices throughout the product lifecycle. For example, there is no reason to have the IIS FTP server or Frontpage Server extensions running if they are not being used. Also, there is no reason to monitor Active Server Pages if your application is completely ASP.NET based.

Most Important IIS Counters

The counters listed in this section include the most important counters for performance and workload characterization. They do not include counters that are not compatible with IIS 6.0.

Note: When monitoring .NET Web-based applications it is recommended to monitor .NET CLR as well. For a list of important counters for .NET CLR monitors, see Chapter 7, ".NET Platform Monitoring."

WWW Service

The Web Service counters help you determine how well the World Wide Web Publishing Service (WWW service) processes requests. The WWW service is a user-mode service. These counters also reflect the processing that occurs in the kernel-mode driver, **HTTP.sys**.

You can configure these counters either per Web site or globally for the entire server by selecting the **_Total** instance.

Counter	Description
Bytes Sent/sec	The rate, in seconds, at which data bytes have been sent by the WWW service
Bytes Received/sec	The rate, in seconds, at which data bytes have been received by the WWW service
Current Connections	The number of active connections to the WWW service
Not Found Errors/sec	The rate, in seconds, at which requests were not satisfied by the server because the requested document was not found
Locked Errors/sec	The rate, in seconds, at which requests were not satisfied because the requested document was locked

Counter	Description
Current ISAPI Extension Requests	The number of ISAPI extension requests that are being processed simultaneously by the WWW service
ISAPI Extension Requests/sec	The rate, in seconds, at which ISAPI extension requests are being processed by the WWW service

WWW Service Cache

The WWW service and FTP service do not share a common cache. Instead, the caches are split into two separate performance objects: one for FTP service and one for the WWW service. WWW service cache counters are designed to monitor server performance only; therefore, you cannot configure them to monitor individual sites.

Counter	Description
Current File Cache Memory Usage	The number of bytes currently used for the user-mode file cache
Current Files Cached	The number of files whose content is currently in the user-mode cache
Current URIs Cached	The number of URI information blocks that are currently stored in the user-mode cache
Current Metadata Cached	The current number of metadata information blocks in the user-mode cache
Kernel: URI Cache Hits/sec	The average number of kernel URI cache hits that are being made per second

ASP.NET

ASP.NET supports the following ASP.NET system performance counters, which aggregate information for all ASP.NET applications on a Web server computer, or, alternatively, apply generally to a system of ASP.NET servers running the same applications.

Note: Not all of these counters are available in all IIS deployments.

Counter	Description
Requests Disconnected	The number of requests that were disconnected because a communication failure occurred.
Requests Queued	The number of requests in the queue waiting to be serviced. If this number increases as the number of client requests increases, the Web server has reached the limit of concurrent requests that it can process. The default maximum for this counter is 5,000 requests. You can change this setting in the computer's Machine.config file.
Requests Rejected	The total number of requests that were not executed because insufficient server resources existed to process them. This counter represents the number of requests that return a 503 HTTP status code, which indicates that the server is too busy.
Errors Total/sec	The average number of errors that occurred per second during the execution of HTTP requests. Includes any parser, compilation, or run-time errors.
Output Cache Turnover Rate	The average number of additions to and removals from the output cache per second. If the turnover is great, the cache is not being used effectively.
Sessions Active	The number of sessions that are active. This counter is supported only with in-memory session state.

Counter	Description
Transactions/sec	The average number of transactions that were started per second.
Transactions Pending	The number of transactions that are in progress.

Active Server Pages

If you are running Active Server Pages (ASP) on your server, the ASP counters can help you determine how well the server or site is responding to ASP requests. The ASP counters are designed to monitor server performance; you cannot monitor individual ASP applications because ASP counters collect global data across the entire WWW service.

Counter	Description
Errors/sec	The average number of errors that occurred per second.
Requests/sec	The average number of requests that were executed per second.
Requests Executing	The number of ASP requests currently executing (for example, the number of active worker threads).
Requests Queued	The number of queued ASP requests that are waiting to be processed. The maximum number for this counter is determined by the metabase property AspRequestQueueMax.
Transactions/sec	The average number of transactions that have been started, per second.

Optimization and Tuning

When performance issues are encountered, tuning and optimization are required to alleviate these issues. In most cases application code optimization is required, but sometimes fixing a poorly tuned environment can dramatically improve performance.

This section lists some possible tuning practices. Some are oriented for the IIS Web server while others are general for any Web server. There are many other tuning practices that might be more effective for your application.

Tuning requires a long and iterative process of testing and analysis. Any configuration change requires careful validation. Before applying any of below practices, you should first validate the relevancy of the configuration to your specific application by understanding the parameter and the workload generated against your server.

- Tune the connection limit. A large number of connections alongside high CPU utilization and high processor queue length indicates a CPU bottleneck. You should either limit maximum connections allowed or increase CPU power.
- Turn off ASP debugging. Verify that both the server and client sides are turned off by setting **AppAllowDebugging** and **AppAllowClientDebug** to **false**.
- Set **AspBufferingOn** to **true** in order to collect the ASP output buffer before it is sent to the client.
- The **AspProcessorThreadMax Metabase** property specifies the maximum number of worker threads per processor that IIS can create. To find out the maximum number of worker threads that IIS allows per ASP process, multiply this value by the number of processors on your server. If you decrease this value, monitor performance to make sure that the lower thread limit does not degrade performance. If it does, increase the value again.
- The **AspRequestQueueMax Metabase** property specifies the maximum number of ASP requests that are permitted in a queue. The default value is 3,000, but the optimal setting depends on the behavior of the application. If the execution time of the request is very short and the time in the queue is short, it is reasonable to decrease this value.

- Verify that the keep-alive state for each TCP connection is enabled (**connection = keep-alive**). If keep-alive connections are turned off, every file requires a new TCP connection. For small files, enabling HTTP Keep-Alives in IIS effectively doubles the number of roundtrips.
- Enable HTTP compression to increase efficiency of bandwidth use.
- Set HTTP expire headers for all images and for HTML so that proxy servers and browsers make fewer calls to the Web server.
- Remove unnecessary file content. Remove unnecessary empty lines, tabs, characters, and so on. Bigger files impact the time it takes to transfer a file over the network.
- Use static files wherever possible in order to reduce processor demand as much as possible.
- Establish Web gardens, which are application pools that can run multiple worker processes.

Part V

Application Server Monitoring

10

WebLogic Monitoring

This chapter describes best practices for WebLogic application server monitoring.

This chapter includes:

Overview

Oracle WebLogic is one of the top J2EE application servers. WebLogic architecture and infrastructure are oriented for performance and scalability and allow deployment of many types of distributed applications such as Web-based applications and Web services. Furthermore, WebLogic's complete implementation of Sun Microsystems Java EE 5.0 specification provides a standard set of APIs for creating distributed Java applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems.

These capabilities, among others, make the WebLogic application server an important environment to become familiar with from the performance perspective. This chapter describes WebLogic application server high level architecture, recommended counters for monitoring, and main tuning-related aspects.

Architecture

WebLogic has different product configurations:

- **WebLogic Server.** Provides the core services and infrastructure for J2EE applications.
- **WebLogic Enterprise.** Consists of WebLogic Server and BEA Tuxedo software.
- **WebLogic Express.** Provides a "lightweight" version, non J2EE, flavor of WebLogic Server.

This chapter focuses on the WebLogic Server.

In order to understand WebLogic architecture and deployment you need to become familiar with WebLogic Server domains.

A WebLogic Server domain is a logically related group of WebLogic Server resources. It includes a special WebLogic Server instance called the Administration Server and additional WebLogic Server instances called Managed Servers. A WebLogic Server instance can be deployed either as an Administration Server or as a Managed Server.

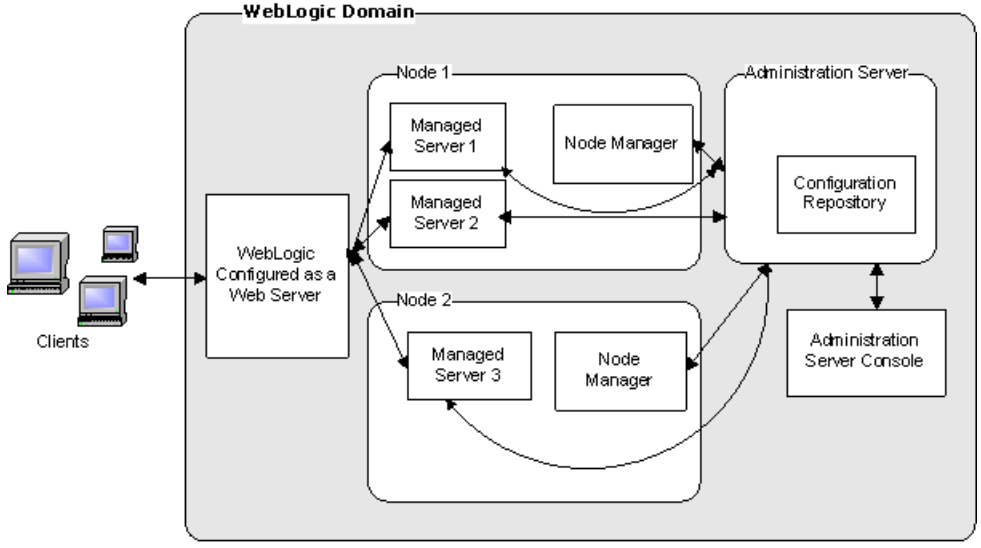
The Administration Server is used solely for the purpose of management and monitoring of the entire domain while the Managed Servers host and execute the application logic deployed in them. Each Managed Server runs under its own Java process, using Oracle JRockit JVM. This also applies to the Administration Server.

In addition to the Administration and Managed Servers, the domain contains additional resources and services that the Managed Servers and applications require. The Node Manager is one such resource. The Node Manager is associated with a machine and not with a logical entity, and allows the domain Administration Server to control the Managed Servers deployed on the machine.

You can use a single WebLogic Server installation to create and run multiple domains, or you can use multiple installations to run a single domain. It is important to understand the considerations that led to the domain configuration of the WebLogic Server in your environment since it often impacts performance and scalability.

Each WebLogic Server can be configured as a Web server utilizing its own HTTP listener, which supports HTTP 1.1. Alternatively, Apache, Microsoft IIS, and Netscape Web servers can also be used. The Web server configuration allows the WebLogic Server to service requests for static HTML content in addition to dynamic content generated by servlets or JSPs.

The following diagram illustrates a WebLogic domain that consist of three Managed Servers which are deployed on two machines/nodes.



Monitoring

The WebLogic Server management system offers management, health, and performance data through a collection of managed beans (MBeans), based on Sun's Java Management Extension (JMX) standard. These MBeans can be queried using either JXM or SNMP. Additionally, the WebLogic Server records information regarding configuration changes and subsystem failures in log files. These log files can be useful for investigating critical failures, but less relevant while applying load.

Recent WebLogic versions include the diagnostics framework, WebLogic Diagnostics Framework (WLDF). WLDF leverages the MBeans mentioned earlier and adds additional capabilities, including the following:

- Capturing diagnostics snapshots that can be used for post-failure analysis
- Archiving data events, log records, and metrics from server instances and applications
- Instrumenting the server and the applications it runs

It is important to thoroughly understand your application's architecture and deployment when checking performance throughout the product lifecycle. This is especially true when monitoring the J2EE application deployed on a WebLogic server. For example, unless your WebLogic server deployment is configured in cluster mode, the Cluster monitor is irrelevant.

The HP SiteScope WebLogic Solution template is the recommended method for monitoring WebLogic using LoadRunner or Performance Center. SiteScope WebLogic Solution is based on the SiteScope WebLogic monitor and JMX monitor with a predefined set of counters for monitoring. It uses the JMX interface for monitoring WebLogic, and therefore requires security access configuration by the WebLogic administrator.

Alternatively, you can use the SiteScope WebLogic monitor for monitoring WebLogic 6.x, 7.x, and 8.x and SiteScope JMX monitor for monitoring WebLogic 9.x or 10.x. The latter requires more manual effort when configuring the desired counters. Detailed instructions are provided in the SiteScope User Guide.

Most Important WebLogic Counters

The list of counters below includes the most important counters for performance and workload characterization. WebLogic exposes many more; in order to monitor them you can select them from the relevant MBean.

The counters are classified by different entities and according to the WebLogic MBeans.

Note: Counters may vary depending on what is installed on the application server.

Server

As work enters a WebLogic Server, it is placed in an execute queue. This work is then assigned to a thread within the queue that performs the work.

The following counters help you assess the server ability to handle the workload and identify whether the Execute Queue or Thread Pool are related to a potential bottleneck.

Counter	Description
MBean: weblogic.management.runtime.ServerRuntimeMBean	
OpenSocketsCurrentCount	The current number of sockets registered for socket muxing on this server.
MBean: weblogic.management.runtime.ExecuteQueueRuntimeMBean	
ExecuteThreadCurrentIdleCount	The number of idle threads assigned to the queue.
ExecuteThreadTotalCount	The total number of execute threads assigned to the queue.
PendingRequestCurrentCount	The number of pending requests in the queue.

Counter	Description
MBean: weblogic.management.runtime.ThreadPoolRuntimeMBean	
ExecuteThreadIdleCount	The number of idle threads in the pool. This count does not include standby threads and stuck threads. The count indicates threads that are ready to pick up new work when it arrives.
ExecuteThreadTotalCount	The total number of threads in the pool.
PendingUserRequestCount	The number of pending user requests in the priority queue. The priority queue contains requests from internal subsystems and users. This is the count of all user requests.
QueueLength	The number of pending requests in the priority queue. This is the total of internal system requests and user requests.
Throughput	The mean number of requests completed per second.
StandbyThreadCount	Returns the number of threads in the standby pool. Surplus threads that are not needed to handle the present workload are designated as standby and added to the standby pool. These threads are activated when more threads are needed.

EJB

Enterprise JavaBeans are the server-side components that encapsulate the business logic. This makes them a major candidate for a performance bottleneck, and therefore are important to monitor.

There are two major types of beans: Session Beans and Message Driven Beans, where the Session Beans can be either Stateful or Stateless.

Counter	Description
MBean: weblogic.management.runtime.EJBCacheRuntimeMBean. Monitors cache counters for Entity Beans and Stateful Beans.	
ActivationCount	Provides a count of the total number of beans from this EJB Home that have been activated.
CacheAccessCount	Provides a count of the total number of attempts to access a bean from the cache. **Note:** The sum of the Cache Hit Count and Cache Miss Count may not add up to the CacheAccessCount in a running server because these metrics are retrieved using multiple calls and the counts could change between the calls.
CachedBeansCurrentCount	Provides a count of the total number of beans from this EJB Home currently in the EJB cache.
CacheMissCount	Provides a count of the total number of times an attempt failed to access a bean from the cache. **Note:** The sum of the Cache Hit Count and Cache Miss Count may not add up to the CacheAccessCount in a running server because these metrics are retrieved using multiple calls and the counts could change between the calls.

Counter	Description
PassivationCount	Provides a count of the total number of beans from this EJB Home that have been passivated.
MBean: weblogic.management.runtime.EJBLockingRuntimeMBean	
LockEntriesCurrentCount	Provides a count of the number of beans currently locked.
LockManagerAccessCount	Provides the total number of attempts to obtain a lock on a bean. This includes attempts to obtain a lock on a bean that is already locked on behalf of the client.
TimeoutTotalCount	Provides the current number of threads that have timed out waiting for a lock on a bean.
WaiterCurrentCount	Provides the current number of threads that have waited for a lock on a bean.
MBean: weblogic.management.runtime.EJBPoolRuntimeMBean Monitors EJB instances for Entity Beans, MessageDriven Beans, and Stateless Beans	
AccessTotalCount	Provides a count of the total number of times an attempt was made to retrieve an instance from the free pool.
BeansInUseCurrentCount	Provides a count of the number of bean instances currently being used from the free pool.
DestroyedTotalCount	Provides a count of the total number of times a bean instance from this pool was destroyed due to a non-application Exception being thrown from it.
MissTotalCount	Provides a count of the total number of times a failed attempt was made to get an instance from the free pool. An attempt to get a bean from the pool fails if there are no available instances in the pool.

Counter	Description
PooledBeansCurrentCount	Provides a count of the current number of available bean instances in the free pool.
TimeoutTotalCount	Provides a count of the total number of threads that have timed out waiting for an available bean instance from the free pool.
WaiterCurrentCount	Provides a count of the number of threads currently waiting for an available bean instance from the free pool.
MBean: weblogic.management.runtime.EJBTransactionRuntimeMBean Monitors transaction counters for Entity Beans, MessageDriven Beans, Stateless Beans, and Stateful Beans	
TransactionsCommittedTotalCount	Provides a count of the total number of transactions that have been committed for this EJB.
TransactionsRolledBackTotalCount	Provides a count of the total number of transactions that have been rolled back for this EJB.
TransactionsTimedOutTotalCount	Provides a count of the total number of transactions that have timed out for this EJB.

Servlet

Counter	Description
MBean: weblogic.management.runtime.ServletRuntimeMBean	
ExecutionTimeAverage	Provides the average amount of time all invocations of the servlet have executed since it was created.

JRockit

These counters are available only if you run a server with JRockit Virtual Machine and are essential for both characterizing the performance of the application, as well as for tuning.

Counter	Description
MBean: weblogic.management.runtime.JRockitRuntimeMBean	
UsedHeap	Indicates the amount (in bytes) of Java heap memory that is currently being used by the VM.
UsedPhysicalMemory	Indicates the amount (in bytes) of physical memory that is currently being used on the host computer. This value reports the memory that is being used by all processes on the computer, and not just by the VM.
TotalNurserySize	Indicates the amount (in bytes) of memory that is currently allocated to the nursery. The nursery is the area of the Java heap that the VM allocates to most objects. Instead of garbage collecting the entire heap, generational garbage collectors focus on the nursery. Because most objects die young, most of the time it is sufficient to garbage collect only the nursery and not the entire heap. If you are not using a generational garbage collector, the nursery size is 0.
AllProcessorsAvgLoad	Displays a snapshot of the average load of all processors in the host computer. If the computer has only one processor, this value is the same as JVM Processor Load. The value is returned as a double, where 1.0 represents 100% load (no idle time) and 0.0 represents 0% load (pure idle time).

Counter	Description
JVMProcessorLoad	Displays a snapshot of the load that the VM is placing on all processors in the host computer. If the host contains multiple processors, the value represents a snapshot of the average load. The value is returned as a double, where 1.0 represents 100% load (no idle time) and 0.0 represents 0% load (pure idle time).
TotalNumberOfThreads	Indicates the number of Java threads (daemon and non-daemon) that are currently running in the VM across all processors.
NumberOfDaemonThreads	Indicates the number of daemon Java threads currently running in the VM across all processors.

JDBC Connection Pool

Java Database Connectivity (JDBC) is a standard Java API for interfacing with database and executing SQL statements.

Database is often a performance bottleneck and it is important to monitor carefully from all angles. The counters below are relevant for the JDBC connection pool. They assist in completing the picture of the database behavior under load.

Counter	Description
MBean: weblogic.management.runtime.JDBCDataSourceRuntimeMBean	
ActiveConnectionsAverageCount	Average number of active connections in this instance of the data source. Active connections are connections in use by an application.
ActiveConnectionsCurrentCount	The number of connections currently in use by applications.

Counter	Description
ConnectionDelayTime	The average amount of time (in milliseconds) that it takes to create a physical connection to the database. The value is calculated as a sum of all the times it took to connect, divided by the total number of connections.
CurrCapacity	The current count of JDBC connections in the connection pool in the data source.
LeakedConnectionCount	The number of leaked connections. A leaked connection is a connection that was reserved from the data source but was not returned to the data source by calling close().
NumAvailable	The number of database connections currently available (not in use) in this data source.
NumUnavailable	The number of database connections that are currently unavailable (in use or being tested by the system) in this instance of the data source.
PrepStmtCacheHitCount	The cumulative, running count of the number of times that statements from the cache were used.
PrepStmtCacheMissCount	The number of times that a statement request could not be satisfied with a statement from the cache.
WaitingForConnectionCurrentCount	The number of connection requests waiting for a database connection.

JMS

WebLogic JMS is an enterprise-class messaging system that is tightly integrated into the WebLogic Server platform.

The following counters are relevant only when your application uses WebLogic JMS. In such case these counters are very useful in determining whether or not the JMS server is a bottleneck.

Counter	Description
MBean: weblogic.management.runtime.JMSRuntimeMBean	
ConnectionsCurrentCount	The current number of connections to WebLogic Server.
MBean: weblogic.management.runtime.JMSServerRuntimeMBean	
BytesCurrentCount	The current number of bytes stored on this JMS server. This number does not include the pending bytes.
BytesPageableCurrentCount	The total number of bytes in all the messages that are currently available for paging out, and have not yet been paged out. The JMS server attempts to keep this number smaller than the "MessageBufferSize" parameter.
BytesPendingCount	The current number of bytes pending (unacknowledged or uncommitted) stored on this JMS server. Pending bytes are over and above the current number of bytes.
BytesReceivedCount	The number of bytes received on this JMS server since the last reset.
DestinationsCurrentCount	The current number of destinations for this JMS server.
MessagesCurrentCount	The current number of messages stored on this JMS server. This number does not include the pending messages.

Counter	Description
MessagesPageableCurrentCount	The number of messages that are currently available for paging in this JMS server but have not yet been paged out.
MessagesPendingCount	The current number of messages pending (unacknowledged or uncommitted) stored on this JMS server. Pending messages are over and above the current number of messages.
MessagesReceivedCount	The number of messages received on this destination since the last reset.
SessionPoolsCurrentCount	The current number of session pools instantiated on this JMS server.

JTA

One of WebLogic's fundamental capabilities is transaction management which provides guarantees that database changes are completed accurately with high integrity.

The following counters are useful when trying to evaluate the workload that the server and application can sustain.

Tip: Evaluate rolled back transactions rates. A rate higher than expected should be investigated by looking at the reason for the rollback, and then correlating it with other counters measured in operating system, application server, database server, and LoadRunner transactions.

Counter	Description
MBean: weblogic.management.runtime.JTARuntimeMBean	
TransactionTotalCount	The total number of transactions processed. This total includes all committed, rolled back, and heuristic transaction completions.
TransactionCommittedTotalCount	The number of committed transactions.
TransactionRolledBackTotalCount	The number of transactions that were rolled back.
TransactionRolledBackTimeoutTotalCount	The number of transactions that were rolled back due to a timeout expiration.
TransactionRolledBackResourceTotalCount	The number of transactions that were rolled back due to a resource error.
TransactionRolledBackAppTotalCount	The number of transactions that were rolled back due to an application error.

Counter	Description
TransactionRolledBackSystemTotalCount	The number of transactions that were rolled back due to an internal system error.
TransactionHeuristicsTotalCount	The number of transactions that completed with a heuristic status.
TransactionAbandonedTotalCount	The number of transaction that were abandoned.
AverageCommitTime	The average amount of time (in milliseconds) it takes the server to commit a transaction.
ActiveTransactionsTotalCount	The total number of active transactions on the server.

Optimization and Tuning

Optimization and tuning are crucial for resolving performance issues. In most cases application code optimization is required, but sometimes fixing a poorly tuned environment can dramatically improve performance.

This section lists a few possible tuning practices. Some are oriented for the WebLogic application server, while others are general for any application server. There are many other tuning practices that can improve the performance of your application.

Tuning requires a long and iterative process of testing and analysis. Any configuration change requires careful validation. Before applying any of the below practices, validate the relevancy of the configuration to your specific application by understanding the parameters and workload generated against your server.

Tune Pool Sizes

Tuning EJB, JDBC, and Thread related pools for their appropriate size increases the server's capacity and it performs better. To tune these pools, you monitor the relevant counters mentioned in the previous section, and look for the amount of waits and LoadRunner transaction response time. Note the optimal response time.

Use the Prepared Statement Cache

The prepared statement cache keeps compiled SQL statements in memory, thus avoiding a round-trip to the database when the same statement is used later.

JVM Tuning

- Examine which collection algorithm fits your application better: concurrent or parallel.
- Determine the optimal heap size.
  - Monitor your application under peak load.
  - Analyze how often collection is taking place. Too frequent collections with shrinking free memory size may require application code optimization.
  - Analyze how long full GC takes. If takes more than 5 seconds, lower the heap size.
  - Analyze the average memory footprint. If heap is 85% free after full GC, its size can be lowered.

Execute Queue

Increase the thread count if the queue length and the CPU are under utilized. This better utilizes the CPU.

General

- Always serve static content such as HTML pages, images, CSS files, JavaScript files using a Web Server. This will reduce the CPU time spent on the application server machine, leaving more time to process other jobs.
- Use WebLogic clustering for scalability and high availability.

11

WebSphere Monitoring

This chapter describes best practices for WebSphere Application Server monitoring.

This chapter includes:

Overview

The IBM WebSphere Application Server is the flagship product in the IBM WebSphere platform. It is one of the top J2EE application servers. WebSphere architecture and infrastructure are oriented for performance and scalability, and allow deployment of many types of distributed applications such as Web-based applications and Web services. Furthermore, WebSphere's complete implementation of the Sun Microsystems Java EE 5.0 specification provides a standard set of APIs for creating distributed Java applications that can access a wide variety of services, such as databases, messaging services, and connections to external enterprise systems.

These capabilities among others make WebSphere Application Server (WAS) an important environment to become familiar with from the performance perspective. This chapter helps you understand WebSphere Application Server high-level architecture, recommended counters for monitoring, and main tuning related aspects.

Architecture

WebSphere Application Server comes in five different editions:

- **WebSphere Application Server Network Deployment.** Delivers near-continuous availability with advanced performance and management capabilities for mission-critical applications.
- **WebSphere Application Server for z/OS.** Provides similar capabilities to the Network Deployment edition, oriented for z/OS and uses, to its advantage, the z/OS Workload Manager.
- **WebSphere Application Server.** Provides Java EE 5 configuration, optimized to ease administration in a scalable, single-server environment.
- **WebSphere Application Server Express**. Provides a scaled down version of the WebSphere Application Server edition.
- **WebSphere Application Server Community Edition.** Provides a lightweight Java EE 5 application server based on open source Apache Geronimo.

Each member of the WebSphere Application Server family uses the same architectural structure with some differences in capabilities, platform compatibility, and licensing.

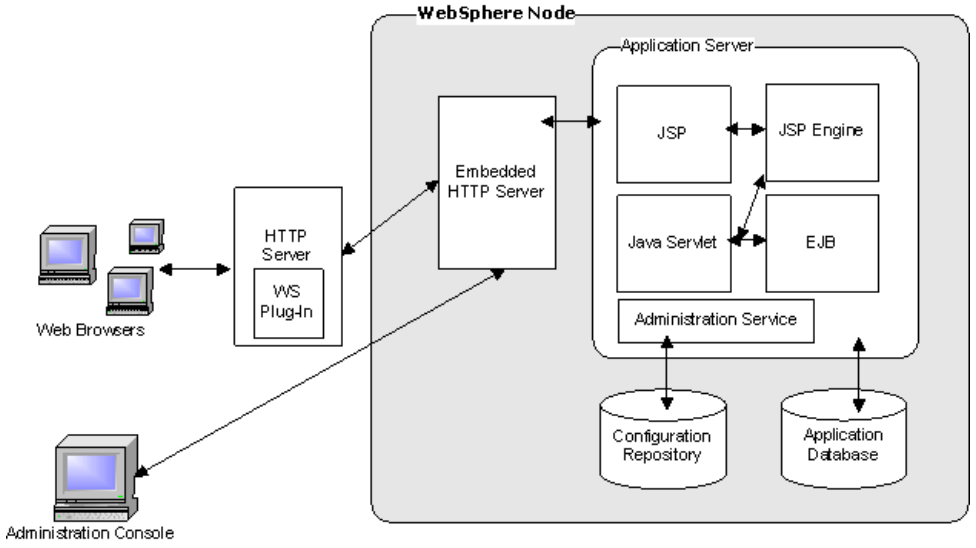
The WebSphere Application Server is organized based on the concept of cells, nodes, and servers. Cells and nodes play an important role when you reach the Network Deployment configuration.

- **Servers.** A server performs the actual code execution. There are several types of servers, depending on the configuration: Application servers and JMS servers. Each server runs on its own JVM.
- **Nodes.** A node is a logical grouping of WebSphere-managed server processes that share common configuration and operational control. A node is generally associated with one physical installation of WebSphere Application Server.
- **Cells.** A cell is a grouping of nodes into a single administrative domain.

A typical WebSphere cell contains software components that may be installed on one node or distributed over multiple nodes for scalability and reliability purposes. These include the following:

- A Web server that provides HTTP services
- A database server for storing application data
- WebSphere Application Server (WAS)

The following diagram illustrates a single WebSphere node architecture:



Monitoring

WebSphere Application Server provides a performance monitoring infrastructure (PMI) which is a server side monitoring infrastructure that offers client-side API. Using PMI you can monitor the overall health and performance of the application server. The performance data is made available via JMX.

Note: PMI is enabled from the WebSphere administrative console.

It is important to thoroughly understand your application's architecture and deployment when checking performance throughout the product lifecycle. This is especially true when monitoring the J2EE application deployed on a WebSphere server. For example, Web Services counters are relevant only if your application has them as well.

HP SiteScope WebSphere Solution template is the recommended method for monitoring WebSphere while using LoadRunner or Performance Center. The Solution template comes with a predefined set of counters for monitoring.

Alternatively, you can use the SiteScope WebSphere monitor. Using this monitor requires manual configuration of the desired counters. Detailed instructions are provided in the SiteScope User Guide.

Most Important WebSphere Counters

The following list of counters includes the most important counters for performance and workload characterization. WebSphere exposes many more; in order to monitor them you can select them while you configure the SiteScope monitor.

The counters below are classified according to the IBM WebSphere classification.

Note: Counters may vary depending on what is installed on the application server.

Enterprise Java Beans

Counter	Key	Description
ReadyCount	beanModule.readyCount	The number of concurrently ready beans (entity and session). This counter was called **concurrent active** in Versions 3.5.5+ and 4.0.
LiveCount	beanModule.concurrentLives	The number of concurrently live beans.
MethodResponseTime	beanModule.avgMethodRt	The average response time (in milliseconds) on the bean methods (home, remote, local).
ActiveMethodCount	beanModule.activeMethods	The number of concurrently active methods - the number of methods called at the same time.

Counter	Key	Description
MessageCount	beanModule.messageCount	The number of messages delivered to the bean onMessage method (message driven beans).
MessageBackoutCount	beanModule.messageBackoutCount	The number of messages that failed to be delivered to the bean onMessage method (message driven beans).
PooledCount	beanModule.poolSize	The number of objects in the pool (entity and stateless).
WaitTime	beanModule.avgSrvSessionWaitTime	The average time taken to obtain a ServerSession from the pool (message driven bean).

JDBC Connection Pool

Counter	Key	Description
Concurrent waiters	connectionPoolModule.concurrentWaiters	The number of threads that are currently waiting for a connection.
Faults	connectionPoolModule.faults	The total number of faults, such as timeouts, in the connection pool.
Percent used	connectionPoolModule.percentUsed	The average percent of the pool that is in use.

Java Virtual Machine (JVM)

Counter	Key	Description
FreeMemory	jvmRuntimeModule.freeMemory	The free memory in the JVM run time.
ProcessCpuUsage	jvmRuntimeModule.cpuUsage	The percentage of CPU usage of the JVM.
UsedMemory	jvmRuntimeModule.usedMemory	The used memory in the JVM run time.

Servlet Session

Counter	Key	Description
ActiveCount	servletSessionsModule.activeSessions	The number of concurrently active sessions. A session is active if the WebSphere Application Server is currently processing a request.
LiveCount	servletSessionsModule.liveSessions	The number of local sessions that are currently cached in memory.

Transaction

Counter	Key	Description
ActiveCount	transactionModule.activeGlobalTrans	The number of concurrently active global transactions.
LocalActiveCount	transactionModule.activeLocalTrans	The number of concurrently active local transactions.
RolledbackCount	transactionModule.globalTransRolledBack	The total number of global transactions rolled back.

Counter	Key	Description
LocalRolledbackCount	transactionModule.localTransRolledBack	The number of local transactions rolled back.
GlobalTimeoutCount	transactionModule.globalTransTimeout	The number of global transactions timed out.
LocalTimeoutCount	transactionModule.localTransTimeout	The number of local transactions timed out.

Thread Pool

Counter	Key	Description
ActiveCount	threadPoolModule.activeThreads	The number of concurrently active threads.
PoolSize	threadPoolModule.poolSize	The average number of threads in pool.
PercentMaxed	threadPoolModule.percentMaxed	The average percentage of the time that all threads are in use.
DeclaredthreadHungCount	threadPoolModule.declaredThreadHung	The number of threads declared hung.

Web Application

Counter	Key	Description
ConcurrentRequests	webAppModule.servlets.concurrentRequests	The number of requests that are concurrently processed.
ServiceTime	webAppModule.servlets.responseTime	The response time (in milliseconds) of a servlet request.

Counter	Key	Description
ConcurrentRequests	webAppModule.url.concurrentRequests	The number of requests processing concurrently for a URI associated with a servlet.
ServiceTime	webAppModule.url.responseTime	The average service response time (in milliseconds) for an URI associated with a servlet.

System

Counter	Key	Description
CPUUsageSinceLast Measurement	systemModule.cpuUtilization	The average system CPU utilization taken over the time interval since the last reading. **Notes:** ➤ Because the first call is required to perform initialization, a value such as 0, which is not valid, will be returned. All subsequent calls return the expected value. ➤ On SMP machines, the value returned is the utilization averaged over all CPUs.
FreeMemory	systemModule.freeMemory	The amount of real free memory available on the system. **Notes:** ➤ Real memory that is not allocated is only a lower bound on available real memory, since many operating systems take some of the otherwise unallocated memory and use it for additional I/O buffering. ➤ The exact amount of buffer memory which can be freed up is dependent on both the platform and the application(s) running on it.

Optimization and Tuning

Optimization and tuning are crucial for resolving perfomance issues. In most cases application code optimization is required, but sometimes fixing a poorly tuned environment can dramatically improve performance.

This section lists a few possible tuning practices. Some are oriented for WebSphere Application Server, while others are general for any application server. There are many other tuning practices that can improve the performance of your application.

Tuning requires a long and iterative process of testing and analysis. Any configuration change requires careful validation. Before applying any of the below practices, validate the relevancy of the configuration to your specific application by understanding the parameters and workload generated against your server.

Tune Pool Sizes

Tuning EJB, JDBC and Thread related pools for their appropriate size increases the server's capacity and it performs better. To tune these pools you monitor the relevant counters (see “Most Important WebSphere Counters” on page 199). In particular, look for the amount of concurrent requests, waits, and LoadRunner transaction response time. The application design needs to be taken into consideration in order to avoid misconfiguration.

Use the Prepared Statement Cache

The prepared statement cache keeps compiled SQL statements in memory, thus avoiding a round-trip to the database when the same statement is used later. The prepared statement cache needs to be sized based on the number of concurrent requests being processed and the design of the application.

JVM Tuning

- Examine which collection algorithm suits your application better: concurrent or parallel.
- Determine the optimal heap size.
  - Monitor your application under peak load.
  - Analyze how often collections take place. Too frequent collections with shrinking free memory size might require application code optimization.
  - Analyze how long full GC takes. If it takes more than 5 seconds, lower the heap size.
  - Analyze the average memory footprint. If the heap is 85% free after a full GC, its size can be lowered.

General

- Always serve static content such as HTML pages, images, CSS files, and JavaScript files using a Web server. This will reduce the CPU time spent on the application server machine, leaving more time to process other jobs.
- Disable functions that are not required. For example, if your application does not use the Web services addressing (WS-Addressing) support, disabling this function can improve performance.
- Ensure that the transaction log is assigned to a fast disk.

Part VI

Database Resource Monitoring

12

Database Resource Monitoring - Introduction

The majority of modern applications are designed to run in multi-tiered architecture, where the functionality of the application is spread across multiple layers or tiers, each typically executing on its own server. These layers usually include, but are not limited to, the following:

- **User Interface.** Bridges the communication between the user and the application.
- **Business Layer.** Associated with all business rules necessary to run an application.
- **Data layer.** Addresses the data required for managing business transactions.

This structure provides certain important benefits such as relatively light client footprint, deployment on the server side only, separation of functionality, no direct access to the database, thus lowering total cost of development and ownership of the application.

With this distributed complexity, each of the layers may cause performance problems. However, more frequently than not, performance engineers find the root of the end user dissatisfaction with performance in slow responses from the database tier.

Databases are always in the process of change - be it data, queries, or some logic. Therefore, it is imperative to ensure optimal performance of the database as this is essential to any data-driven application of today.

There are many factors affecting overall application performance that originated on the database side, such as:

- Poor database design during the application development
- Poor standards followed in table design
- Poor indexing of databases
- Poor partitioned data across tables
- Poor logic used in queries
- Inappropriately stored procedures
- Poorly configured storage hardware
- Database server machines dedicated to multiple applications

13

Oracle Monitoring

This chapter describes best practices for Oracle monitoring.

This chapter includes:

Overview

The Oracle database is a relational database management system (RDBMS) produced by the Oracle Corporation. The Oracle database is rich with features that contribute to its high availability, scalability, performance, manageability, and security. These features make Oracle an enterprise class RDBMS and one of the top leaders in this realm.

The Oracle database has comprehensive support for application development owing to different capabilities and features. Oracle also offers data access methods for both Java and .NET.

The Oracle database comes in several editions, each targeted to different a scale of usage:

- **Standard Edition (SE).** Contains base database functionality. Oriented typically for servers running one to four CPUs. If the number of CPUs exceeds 4 CPUs, the user must convert to an Enterprise license. SE has no memory limits, and can utilize clustering with Oracle RAC.
- **Enterprise Edition (EE).** Extends the 'Standard Edition', especially in the areas of performance and security. Oriented for servers running 4 or more CPUs. EE has no memory limits, and can utilize clustering using Oracle RAC software.
- **Standard Edition One.** Introduced with Oracle 10g, has some feature-restrictions comparing to the 'Standard Edition'. Oriented for use on systems with one or two CPUs. It has no memory limitations.
- **Express Edition ('Oracle Database XE').** Introduced in 2005, it is free to distribute on Windows and Linux platforms. It has a footprint of only 150 MB and is restricted to the use of a single CPU and a maximum of 4 GB of user data. Although it can be installed on a server with any amount of memory, it is limited to using 1 GB at most.
- **Oracle Database Lite.** Intended to run on mobile devices. The database, partially located on the mobile device, can synchronize with a server-based installation.

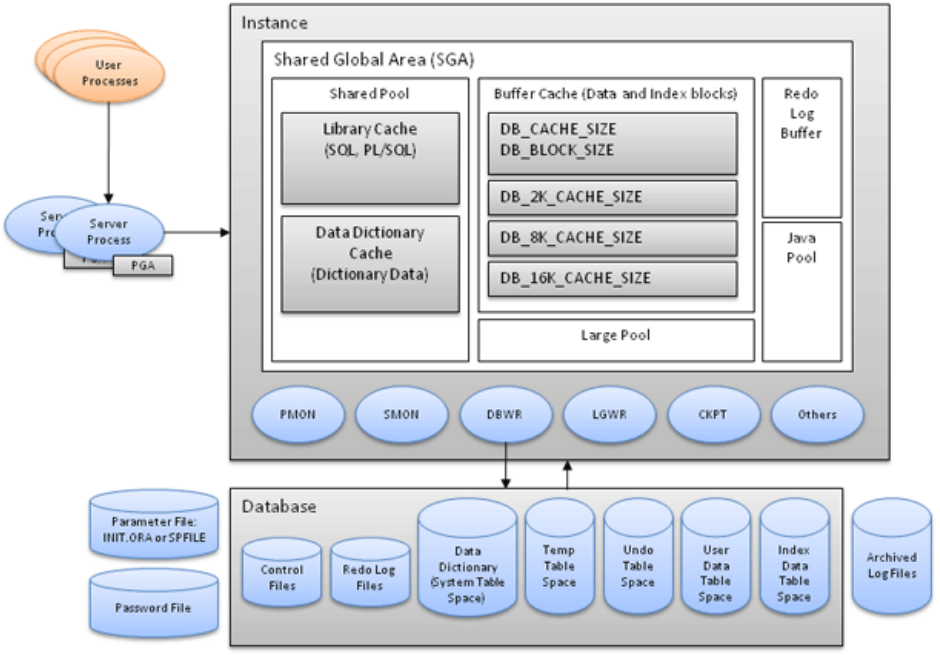
It is a known fact that the database tier has a great deal of influence on application performance. Oracle, as one of the top players in the database field, is an important environment to get familiar with from a performance perspective. This chapter helps you understand the Oracle database's high-level architecture, learn about its monitoring capabilities. It also lists the most important counters for monitoring and describes some tuning related practices.

Architecture

The Oracle database consists of an instance and data storage. The instance is a set of operating system processes and memory structures that interact with the storage. The memory structure is called the System Global Area (SGA) and storage is logically stored as tablespaces and physically as data files. Tablespaces can contain various types of memory segments. Segments, in turn, comprise one or more extents. Extents comprise groups of contiguous data blocks and data blocks form the basic units of data storage. At the physical level, data files comprise one or more data blocks, where the block size can vary from one data file to another.

Oracle database management tracks its computer data storage with the help of information stored in the SYSTEM tablespace. The SYSTEM tablespace contains the data dictionary - and often (by default) indexes and clusters. A data dictionary consists of a special collection of tables that contain information about all the user objects in the database.

The following diagram illustrates Oracle database architecture. It displays the different memory structures on the instance level as well as the data files on the storage level.



Each Oracle instance uses a **System Global Area (SGA)**, which is a shared memory area, to store its data and control information. The instance allocates itself an SGA when it starts, and deallocates it at shutdown time. The information in the SGA consists of the following elements, each of which has a fixed size, established at instance startup:

- **Buffer cache.** Stores the most recently used data blocks. This helps Oracle reduce I/O and improve performance as new requests for the same data are fetched from the buffer cache and not from the disk.
- **Redo log buffer.** Stores redo entries, that is, a log of changes made to the database. This helps Oracle recover the instance in case of system failure.

- **Shared pool.** Stores shared memory structures such as shared SQL areas in the library cache and internal information in the data dictionary. An insufficient amount of memory allocated to the shared pool can cause performance degradation.
  - **Library cache.** Stores shared SQL, caching the parse tree and the execution plan for every unique SQL statement. This reduces the amount of memory needed and reduces the processing time used for parsing and execution planning.
  - **Data Dictionary cache.** Stores information such as user information, privileges, table names, datatypes, and so on. The data dictionary helps Oracle parse SQL statements. Performance bottlenecks in the data dictionary affect all Oracle users.

The **Program Global Area (PGA)** is a server-side process serving a user process running on the client machine. The PGA memory area contains data and control information for Oracle's server processes. The PGA holds information regarding the user session, the session variables, sorts, bind variables, and so on.

Oracle typically relies on a group of processes, running simultaneously in the background and interacting, to monitor the database and enhance its performance. The following processes are part of a longer list of processes running on the instance level:

- **Database writer processes (DBWR).** Responsible for writing data to the disk.
- **Log-writer process (LGWR).** Responsible for writing data to the log.
- **System monitor process (SMON).** Responsible for instance recovery, deallocation of temporary segments, and merging free space areas.
- **Process monitor (PMON).** Responsible for cleaning up after failed processes.
- **Checkpoint process (CKPT).** Responsible for signaling about a checkpoint and updating relevant files that a checkpoint has occurred.

The Java Pool is relevant only when Java code is running on the instance level and the Large Pool is optional. In the event that the Large Pool is used, it comes to ease the overhead on the Shared Pool by storing some of the information that the Shared Pool stores by default.

Oracle architecture comes to provide the optimal performance possible throughout reducing I/O operations to their minimum. Performance monitoring and tuning should validate whether the configuration on your deployment indeed leverages these capabilities.

Monitoring

Oracle provides several tools and utilities for performance monitoring and tuning.

- **Automated Database Diagnostics Monitor (ADDM).** Allows an Oracle database to diagnose itself and determine how potential problems could be resolved. ADDM runs automatically after each Automatic Workload Repository (AWR) statistics capture, making the performance diagnostic data readily available. Since AWR captures occur on a regular basis, this ensures that the database diagnoses its performance, and detects the root cause. ADDM considers the following issues as problems:
  - **CPU bottlenecks.** Is the system CPU bound by Oracle or some other application?
  - **Undersized memory structures.** Are the Oracle memory structures, such as the SGA, PGA, and buffer cache, adequately sized?
  - **I/O capacity issues.** Is the I/O subsystem performing as expected?
  - **High load SQL statements.** Are there any SQL statements that are consuming excessive system resources?
  - **High load PL/SQL execution and compilation**, as well as high load Java usage.
  - **RAC specific issues.** What are the global cache hot blocks and objects; are there any interconnect latency issues?

- **Sub-optimal use of Oracle by the application.** Are there problems with poor connection management, excessive parsing, or application level lock contention?
- **Database configuration issues.** Is there evidence of incorrect sizing of log files, archiving issues, excessive checkpoints, or sub-optimal parameter settings?
- **Concurrency issues.** Are there buffer busy problems?
- **Hot objects and top SQL** for various problem areas.

This makes ADDM and AWR reports a very meaningful tool for identifying performance issues and a starting point for tuning.

- **Oracle Enterprise Manager.** Provides a set of systems management tools for managing the Oracle environment. It has tools to monitor the Oracle environment and automate tasks.
- **SQL Trace.** Provides performance information on individual SQL statements. It generates the following statistics for each statement:
  - Parse, execute, and fetch counts
  - CPU and elapsed times
  - Physical reads and logical reads
  - Number of rows processed
  - Misses on the library cache
  - Username under which each parse occurred
  - Each commit and rollback
- **TKProf.** A utility used to format SQL Trace output into human readable format. It is very helpful during the effort of SQL statements tuning. It can also be used for determining execution plans for SQL statements and for creating an SQL script that stores the statistics in the database.

Oracle stores information relevant for monitoring in different statistics tables. These tables are also used by the Oracle SQL statement optimizer. For example:

- Session statistics, V$SESSTAT
- System statistics, V$SYSSTAT
- V$LATCH, V$BUFFER_POOL_STATISTICS

HP monitoring solutions leverage the data in these tables, allowing accessing the data while running a load test. It is recommended to use the HP SiteScope Oracle Database Solution that has recommended built-in counters.

Most Important Oracle Counters

Counter	Description
sorts (disk) (V$SYSSTAT 1/sid) (absolute)	Number of sort operations that required at least one disk write. Sorts that require I/O to disk are quite resource intensive. You might want to increase the size of the initialization parameter SORT_AREA_SIZE.
sorts (memory) (V$SYSSTAT 1/sid) (absolute)	Number of sort operations that were performed completely in memory and did not require any disk writes. You cannot do much better than memory sorts, except maybe no sorts at all. Sorting is usually caused by selection criteria specifications within table join SQL operations.
db block gets (V$SYSSTAT 1/sid) (absolute)	Number of blocks accessed in buffer cache for INSERT, UPDATE, DELETE, and SELECT FOR UPDATE. Represent block logical reads (from cache). The logical reads ALWAYS include the physical reads. Low number of physical reads is preferable.

Counter	Description
consistent gets (V$SYSSTAT 1/sid) (absolute)	Number of blocks accessed in buffer cache for normal queries (SELECTs without for update clause). Represent block logical reads (from cache). The logical reads ALWAYS include the physical reads. Low number of physical reads is preferable.
physical reads (V$SYSSTAT 1/sid) (absolute)	Total number of data blocks read from disk. This number equals the value of **physical reads direct** plus all reads into buffer cache. Low number of physical reads is preferable. This number must be compared to logical reads to calculate cache hit ratio. Logical reads is the sum of database block gets and consistent gets.
physical writes (V$SYSSTAT 1/sid) (absolute)	Total number of data blocks written to disk. This number equals the value of **physical writes direct** plus all writes from buffer cache.
redo writes (V$SYSSTAT 1/sid) (absolute)	Total number of writes by LGWR to the redo log files. **redo blocks written** divided by this statistic equals the number of blocks per write.
redo entries (V$SYSSTAT 1/sid) (absolute)	Redo entries contain the information necessary to reconstruct, or redo, changes made to the database by INSERT, UPDATE, DELETE, CREATE, ALTER, or DROP operations. Redo entries are used for database recovery, if necessary. Redo entries -> successful redo writes. Ratio **Redo buffer allocation retries** / **Redo entries** should be less than 1%.
redo buffer allocation retries (V$SYSSTAT 1/sid) (absolute)	Total number of retries necessary to allocate space in the redo buffer. Retries are needed either because the redo writer has fallen behind or because an event such as a log switch is occurring. Redo buffer allocation retries -> failed redo writes. Ratio **Redo buffer allocation retries** / **Redo entries** should be less than 1%.

Counter	Description
redo log space requests (V$SYSSTAT 1/sid) (absolute)	Number of times the active log file is full and Oracle must wait for disk space to be allocated for the redo log entries. Such space is created by performing a log switch. Log files that are small in relation to the size of the SGA or the commit rate of the work load can cause problems. When the log switch occurs, Oracle must ensure that all committed dirty buffers are written to disk before switching to a new log file. If you have a large SGA full of dirty buffers and small redo log files, a log switch must wait for DBWR to write dirty buffers to disk before continuing. Also examine the log file space and log file space switch wait events in V$SESSION_WAIT
parse count (hard) (V$SYSSTAT 1/sid) (absolute)	Total number of parse calls (real parses). A hard parse is a very expensive operation in terms of memory use, because it requires Oracle to allocate a workheap and other memory structures and then build a parse tree. Should be minimized. The ratio of Hard Parse to Total should be less than 20%.
parse count (total) (V$SYSSTAT 1/sid) (absolute)	Total number of parse calls (hard and soft). A soft parse is a check on an object already in the shared pool, to verify that the permissions on the underlying object have not changed. The ratio of Hard Parse to Total should be less than 20%.
parse time cpu (V$SYSSTAT 1/sid) (absolute)	Total CPU time used for parsing (hard and soft) in 10s of milliseconds.
parse time elapsed (V$SYSSTAT 1/sid) (absolute)	Total elapsed time for parsing, in tens of milliseconds. Subtract **parse time cpu** from this statistic to determine the total waiting time for parse resources.

Counter	Description
CPU used by this session (V$SYSSTAT 1/sid) (absolute)	Amount of CPU time (in tens of milliseconds) used by a session from the time a user call starts until it ends. If a user call completes within 10 milliseconds, the start- and end-user call times are the same for purposes of this statistic, and 0 milliseconds are added.
bytes sent via SQL*Net to client (V$SYSSTAT 1/sid) (absolute)	Total number of bytes sent to the client from the foreground processes. Gives a general indication regarding the amount of data transferred over the net.
bytes received via SQL*Net from client (V$SYSSTAT 1/sid) (absolute)	Total number of bytes received from the client over Oracle Net Services. Gives a general indication regarding the amount of data transferred over the net.
logons current (V$SYSSTAT 1/sid) (absolute)	Total number of current logons. Useful only in V$SYSSTAT.

In addition to the counters mentioned above, it is recommended to monitor relevant tablespace usage. In the case of less than 2% free space in any of them, the tablespace size should be increased.

Optimization and Tuning

When performance issues are encountered, optimization and tuning are required to alleviate the issues. In most cases, application-code optimization is required, but sometimes fixing poorly tuned environment can dramatically improve performance.

This section lists a few possible tuning practices. Some are oriented for the Oracle database, while others are general for any database server, and the rest for any server. There are many other tuning practices that might be more effective for your application.

Tuning requires a long and iterative process of testing and analysis. Any configuration change requires careful validation. Before applying any of the practices mentioned below, you should first validate the relevancy of the configuration to your specific application by understanding the parameters and workload generated against your server.

- Make sure Oracle Cost Based Optimizer is running.
- Gather optimizer statistics on a regular basis.
- Tune SQL statements:
  - Identify problematic SQL statements (that is, long performing SQL statements)
  - Review Oracle optimizer statistics (make sure the cost-based optimizer is running and statistics are up to date)
  - Review execution plan
  - Restructure SQL statement (if necessary)
  - Restructure index (if necessary)
  - Maintain execution plans over time
- Use bind variables in your SQL statements. This will reduce the amount of cursors stored in the shared pool.
- Use indexes carefully. Not every column should be indexed, only those that are accessed more using queries.

- Assist the SQL optimizer by using optimization hints whenever necessary. This should be done after analyzing the SQL statement performance.
- Tune the memory structure size. The size of the Shared Pool, Buffer Cache, and other memory structures is critical for the performance of the database.
  - Run a typical workload against the application
  - Monitor waits, buffer hit ratio, system swapping and paging, and so on
  - The following list includes the most important parameters, among others, that should be tuned:

Parameter	Description
db_cache_size	Determines the size of the buffer cache in the SGA.
db_keep_cache_size	This is where the objects are always present when they are loaded. The objects that qualify for this cache are those which are very frequently accessed and which have to be retained in memory, for example, small frequently used lookup tables. This cache is a subset of default cache defined by parameter DB_CACHE_SIZE. For any database, the DB_CACHE_SIZE must be set.
shared_pool_size	Determines the size of the shared pool.
pga_aggregate_target	Specifies the target aggregate PGA memory available to all server processes attached to the instance.
log_buffer	Determines the size of the redo log buffer.
query_rewrite_enabled	Determines whether Oracle rewrites an SQL statement before it is executed.
cursor_sharing	Determines what kind of SQL statements can share the same cursors.

Parameter	Description
db_file_multiblock_read_count	One of the parameters used to minimize I/O during full table scans. Specifies the maximum number of blocks read in one I/O operation during a sequential scan.
hash_multiblock_io_count	Specifies how many sequential blocks a hash join reads and writes in one I/O.

- To avoid I/O operations, you should aim for a high buffer-cache hit ratio. This should be higher than 80 in an OLTP environment. 99 is the best value.
- The Dictionary cache hit ratio should be around 90%. Entries for **dc_table_grants**, **d_user_grants**, and **dc_users** should be under 5% each in the **MISS RATE %** column.
- Monitor Sorts refer to sorts in memory vs. sorts in disk. The ratio between disk and memory should be less than .10.
- Reduce database contention to a minimum. Study the amount of locks and latches and eliminate whenever possible.
- Use the HP Oracle Database machine for complex large-scale data warehousing workloads.

14

MS SQL Server Monitoring

This chapter describes best practices for Microsoft SQL Server monitoring.

This chapter includes:

Overview

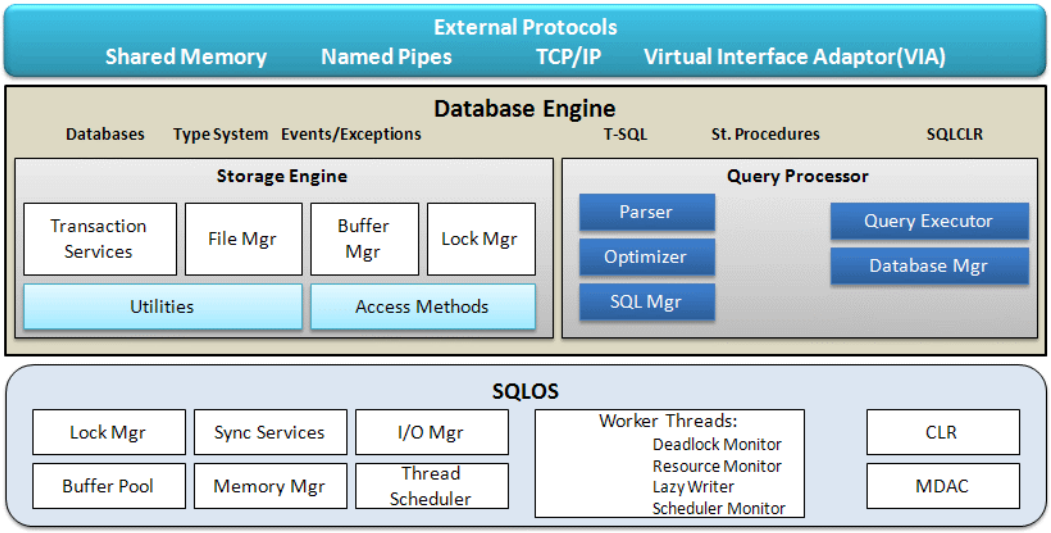
Microsoft SQL Server is one of the most widely used database systems. It has grown from handling small departmental tasks to serving up the largest databases on the planet. No longer a simple "database", Microsoft SQL Server is now a complete data architecture solution capable of handling the data storage and manipulation needs of any organization. Organizations can use this solution to store and manage many types of data, including XML, email, time/calendar, file, document, geospatial, and so on, while providing a rich set of services to interact with the data: search, query, data analysis, reporting, data integration, and robust synchronization. Developers can write applications that access SQL Server from a server to a desktop or mobile device using a variety of technologies, whether Microsoft based or third party vendors.

SQL Server is available in many editions to help meet the needs of any organization. From Express and Compact to Workgroup to Standard and Enterprise, each edition delivers sets of features targeted to specific needs while maintaining the same level of functionality for developers and end users.

It used to be said that SQL Server works great right out of the box and performance is never an issue. However, the advent of cheaper hardware and the explosion of data is pushing more users against the limits of the out-of-the-box performance of SQL Server. It is the job of the performance engineer to find these problems by using various monitoring techniques. In the SQL Server world, the Enterprise and Standard editions are of most interest.

Architecture

The performance behavior of almost every component of SQL Server has been exposed via specific counters which are added to regular Windows objects and counters once SQL Server is installed. However, you will usually start from monitoring Windows system resources such as CPU utilization, disk activity, memory management, and network bandwidth (see Chapter 3, “Windows Monitoring”).



The reason to monitor these resource domains is that they represent the major hardware components of a server, and each component is involved in servicing user requests. The timely performance of these components is directly related to overall perceived application performance. Therefore, a problem with one or more of these four areas is likely to result in user complaints. SQL Server relies heavily on CPU performance, available memory, and disk throughput, whereas the client performance depends heavily on network performance. Any processor which is consistently busy for 90 percent of the time or more will result in work requests queueing, and performance will likely suffer.

In addition, SQL Server can be very demanding on memory, and performance can really suffer if physical memory becomes exhausted, when typically Windows is forced to use the page file. Disk is almost certainly the slowest component because of its mechanical nature. SQL Server's need to retrieve data from disk often means any delays at the disk I/O will impact overall performance. Finally, your database could be performing perfectly well, but if there is latency in the network or if packet loss is high, forcing retransmissions, your server's brilliant speed will be non-existent in the eyes of the end user.

Related Windows Counters

When monitoring system resources of a machine with SQL Server installed, there are some important counters to be tracked, including additional recommendations:

- **CPU.** Adding new physical processors is not an easy task, hence it is important to make sure that all CPU units are equally engaged under load. Watch the following counters:

Note: For complete details about these counters, see "Processor - Most Important Counters" on page 48.

  - **% Processor Time.** Measures individual processor time to ensure load balancing between CPUs.
  - **Processor Queue Length.** If this counter regularly exceeds the recommended maximum, but the CPU utilization is not correspondingly as high (which is typical), then consider reducing the SQL Server **maximum worker threads** configuration setting. Doing this forces thread pooling to start or to take greater advantage of it.

- **Context Switches/sec.** There are two ways to lower this value:
  - **Affinity mask.** Under heavy load, specifying which processor runs which thread improves performance by reducing the number of times the processor cache needs to be reloaded. Sometimes excluding some processors from SQL Server's reach helps improve handling operating system requests.
  - **Lightweight pooling.** When using this SQL Server option, the database turns to a fiber-based model rather than a default thread-based model. Fibers are scheduled by the database server instead of the operating system, so there is less CPU load.

- **Memory.** SQL Server manages its memory dynamically, requesting or releasing it from the operating system. Make sure that appropriate dynamic options are selected and that the maximum memory available for the database is close to the physical highest level.

  Watch the following counters:

Note: For complete details about these counters, see "Memory - Most Important Counters" on page 55.

  - **Available Bytes**
  - **Pages/sec.** Indicates the number of times disk I/O and/or memory outside SQL Server's allocated range is accessed. This value should ideally be close to 0 with possible spikes for backups and restore.
  - **Page Faults/sec**

➤ **Disk.** Database probably has the most I/O intensive operation of all application tiers, so monitoring disk activity is critical.

Watch the following counters:

Note: For complete details about these counters, see "I/O - Most Important Counters" on page 66.

➤ **% Disk Time.** The percentage of time spent on read/write functions. You monitor physical disk counters for single disk volume and logical disk counters for volumes spanning multiple disks. If this value exceeds 55%, this is a clear indication of I/O bottleneck. In this case, you may also want to drill down to **%Disk Read Time** and **%Disk Write Time** counters. The sum of these is the value for this counter. Possible tuning may include adding more and faster disks, getting more cache to the disk controller, defragmentation, and reconfiguration of RAID devices.

➤ **Avg. Disk Queue.** Shows actual queue length for a specific disk, although this counter may be sort of arbitrary in the age of storage area networks (SAN). You monitor physical disk counters for single disk volume and logical disk counters for volumes spanning multiple disks. Do not add the **_Total** counter, as this can generalize the result and mask problems which could lead to you making false assumptions about disk performance.

Tip: It is a very good practice to separate SQL Server data and log files on different disks as they have different I/O patterns. It is also recommended to separate system and user databases onto different disks.

➤ **Network.** Some applications are designed to be very "chatty" when there is a lot of data is sent over the network.

Watch the following counter:

Note: For complete details, see “Network - Most Important Counters” on page 73.

➤ **Bytes Total/sec.** Along with more specific Bytes Received/sec and Bytes Sent/sec counters, shows actual network card throughput. Possible tuning may include adding more and faster network cards, using full duplex option of the card. Reconfigure database settings to remove all unnecessary protocols, leaving TCP/IP as the primary one on both server and client.

Most Important SQL Server Counters

SQL Server performance architecture follows Microsoft’s approach implemented in the Windows operating system and .NET Framework. As such it is organized around objects, instances, and counters (see details about Windows architecture on page 46). An object is any SQL Server resource, such as an SQL Server lock or Windows XP process. Each object contains one or more counters that determine various aspects of the objects to monitor. Some objects have several instances if multiple resources of a given type exist on the computer. Counters for the default instance appear in the format SQLServer:<object name>. Counters for named instances appear in the format MSSQL$<instance name>:<counter name> or SQLAgent$<instance name>:<counter name>.

There are quite a few SQL performance objects including:

➤ 20 objects for the SQLServer engine itself

➤ Three objects for Service Broker

➤ Four objects for SQLAgent

➤ Five objects for SQL Replication

The following table lists database engine counters:

	Counter	Description
CPU	SQL Compilations/sec	Indicates the number of times compilations occurred, per second
	SQL Re-Compilations/sec	Indicates the number of times re-compilations occurred, per second
	Batch Requests/Sec	Indicates the number of Transact-SQL command batches received per second
Memory	Total Pages	Indicates the number of pages in the buffer pool
	Target Pages	Indicates the ideal number of pages in the buffer pool
	Total Server Memory (KB)	Indicates the amount of memory the KB SQL Server is currently using
	Target Server Memory (KB)	Indicates the amount of memory the KB SQL Server needs to operate efficiently
	Buffer cache hit ratio	Indicates the percentage of pages that were found in the memory
	Page Life Expectancy	Indicates the number of seconds a page will stay in the buffer pool without reference
	Stolen Pages	Indicates the number of pages used for miscellaneous server purposes (including procedure cache)
	Cache hit ratio	Indicates the ratio between cache hits and lookups
	Memory Grants Pending	Indicates the total number of processes waiting for a workspace memory grant
	Checkpoint pages/sec	Indicates the number of pages flushed to disk per second by a checkpoint or other operation that requires all dirty pages to be flushed
	Lazy writes/sec	Indicates the number of buffers written per second by the buffer manager's lazy writer

	Counter	Description
Disk	Full Scans/sec	Indicates the number of unrestricted full scans per second
	Page Splits/sec	Indicates the number of page splits per second that occur as a result of overflowing index pages
	Temp Tables Creation Rate	Indicates the number of temporary tables/table variables created per second
Locks	Average Wait Time (ms)	Indicates the average amount of wait time (in milliseconds) for each lock request that resulted in a wait

CPU-Related Counters

If **sqlserver.exe** utilizes most CPU capacity, this may point to issues inside SQL Server. In addition to the Windows counters explained in "Related Windows Counters" on page 228, these issues can be revealed using the following counters:

SQL Compilations/sec

Official Name	SQLServer:SQL Statistics\SQL Compilations/sec
Counter Type	Interval difference counter (rate/second)
Description	The number of times per second that SQL Server compilations occurred.
Usage Notes	A common cause of excessive CPU utilization, which could be caused by schema problems or low memory conditions, is query execution plan compilation and re-compilation. When compiled, plans should remain in memory — unless there is excessive memory pressure that may cause plans to be dropped from the cache.
Performance	Under steady conditions, you should expect to see at least 90 percent plan re-use.
Threshold	Warning when over 10%
Related Measurements	➤ SQL Re-Compilations/sec

SQL Re-Compilations/sec

Official Name	SQLServer:SQL Statistics\SQL Re-Compilations/sec
Counter Type	Interval difference counter (rate/second)
Description	The number of times per second that SQL Server re-compilations occurred.
Usage Notes	If only ad hoc T-SQL is used or queries are not parameterized properly, SQL Server may not re-use any plans, or cause plan compilation for every query.
Performance	A recompile can cause deadlocks and compile locks that are not compatible with any locking type.
Threshold	Warning when over 10% of SQL Compilations/sec
Related Measurements	➤ SQL Compilations/sec

Batch Requests/sec

Official Name	SQLServer:SQL Statistics\Batch Requests/sec
Counter Type	Interval difference counter (rate/second)
Description	The number of Transact-SQL command batches received per second.
Usage Notes	Shows how busy SQL Server's CPUs are.
Performance	If this counter goes above threshold this could mean that if you are not already experiencing a CPU bottleneck, you may very well experience one soon. Of course, this is a relative number, depending on hardware capabilities.
Threshold	Warning when over 1000
Related Measurements	N/A

Note: Some performance engineers monitor *SQLServer:Databases\ Transaction/Sec: _Total* counter which measures activities taken inside transactions only—not all the activities like the *Batch Requests/sec* counter does.

Memory-Related Counters

SQL Server performance and stability are entirely dependent on sufficient available memory. A memory shortage often results in Windows serving the virtual address space from the paging file, which usually has an immediate and very apparent impact on performance.

In addition to the Windows counters explained in "Related Windows Counters" on page 228, use the following counters to monitor memory-related issues:

Total Pages

Official Name	SQLServer:Buffer Manager\Total Pages.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of pages in the buffer pool (includes database, free, and stolen pages).
Usage Notes	Shows the total number of pages acquired by SQL Server from Windows operating system.
Performance	Indicates other processes running on the machine, taking physical memory from SQL Server.
Threshold	See tip on page 236
Related Measurements	➤ Target Pages

Target Pages

Official Name	SQLServer:Buffer Manager\Target Pages.
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The ideal number of pages in the buffer pool.
Usage Notes	Shows the total number of pages required by SQL Server to process requests.
Performance	N/A
Threshold	See tip below.
Related Measurements	➤ Total Pages

Tip: If the Target Pages and Total Pages values are the same, the SQL Server has sufficient memory. If the Target is greater than the Total, it is usually due to another Windows process which is preventing the SQL Server from acquiring as much memory as it would like in order to operate.

Total Server Memory (KB)

Official Name	SQLServer:Memory Manager\Total Server Memory (KB).
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The amount of memory in kilobytes that SQL Server is currently using.
Usage Notes	Shows the total number of physical memory acquired by SQL Server from the Windows operating system.
Performance	Should be less than total amount of memory on the machine.
Threshold	See tip on page 237
Related Measurements	➤ Target Server Memory (KB)

Target Server Memory (KB)

Official Name	SQLServer:Memory Manager\Target Server Memory (KB).
Counter Type	Instantaneous (sampled once during each measurement period).
Description	How much memory the SQL Server would like to have in order to operate efficiently.
Usage Notes	Shows the total amount of memory required by SQL Server to process requests.
Performance	N/A
Threshold	See tip below.
Related Measurements	➤ Total Server Memory (KB)

Tip: If the value of Total Server Memory (KB) is less than the value of Target Server Memory (KB), then SQL Server does not have enough memory to run efficiently. Consider adding more physical memory.

Buffer cache hit ratio

Official Name	SQLServer:Buffer Manager\Buffer cache hit ratio.
Counter Type	Interval (% Busy).
Description	The percentage of pages that were found in the memory.
Usage Notes	The ratio is the total number of cache hits divided by the total number of cache lookups over the last few thousand page accesses.
Performance	If data pages are not found in the buffer, the SQL Server must read them into the buffer from disk. This is usually a slow process because of disk latency and seek times. Hence, if, after configuring the buffer pool to at least 98% of this counter value, performance is still poor, consider adding physical memory.
Threshold	The higher the value the better. Preferred around the 90% mark.
Related Measurements	N/A

Page Life Expectancy

Official Name	SQLServer:Buffer Manager\Page Life Expectancy.
Counter Type	Average.
Description	The number of seconds a page will stay in the buffer pool without references.
Usage Notes	The longer the page life expectancy, the healthier the server looks from a memory perspective.
Performance	Clear indicator of low memory on the server.
Threshold	Problematic if less than 300 seconds.
Related Measurements	N/A

Stolen Pages

Official Name	SQLServer:Buffer Manager\Stolen Pages
Counter Type	Instantaneous (sampled once during each measurement period).
Description	The number of pages used for miscellaneous server purposes (including procedure cache).
Usage Notes	Stolen pages are those pages in memory which are stolen by another process on the SQL Server machine.
Performance	High quantities of stolen pages are a clear indicator of low memory on the server.
Threshold	N/A
Related Measurements	➤ Total Pages

Cache Hit Ratio

Official Name	SQLServer:Plan Cache\Cache Hit Ratio.
Counter Type	Interval (% Busy).
Description	Ratio between cache hits and lookups.
Usage Notes	Percentage of time that the record was found in cache. **Note:** In SQL Server 2000, this counter was found under Cache Manager object.
Performance	This counter is a good indicator of the caching mechanism in SQL Server.
Threshold	Should be around 99%. A value of 90% should generate a warning.
Related Measurements	N/A

Memory Grants Pending

Official Name	SQLServer:Memory Manager\Memory Grants Pending
Counter Type	Instantaneous (sampled once during each measurement period).
Description	Indicates the total number of processes waiting for a workspace memory grant.
Usage Notes	This is effectively a queue of processes awaiting a memory grant.
Performance	If there are any processes queuing waiting for memory, you should expect degraded performance. The ideal situation for a healthy server is no outstanding memory grants.
Threshold	Problematic if not 0.
Related Measurements	N/A

Checkpoint Pages/sec

Official Name	SQLServer:Buffer Manager\Checkpoint pages/sec.
Counter Type	Interval difference counter (rate/second).
Description	Indicates the number of pages flushed to disk per second by a checkpoint or other operation that requires all dirty pages to be flushed.
Usage Notes	The checkpoint operation is performed by SQL Server and requires all dirty pages to be written to disk.
Performance	The checkpoint process is expensive in terms of disk I/O. When a server is running low on memory, the checkpoint process will occur more frequently than usual as SQL Server attempts to create space in the buffer pool. Clear indicator of low memory on the server.
Threshold	Problematic if consistent high values are observed over period of time.
Related Measurements	N/A

Lazy Writes/sec

Official Name	SQLServer:Buffer Manager\Lazy Writes/sec
Counter Type	Interval difference counter (rate/second)
Description	Indicates the number of buffers written per second by the buffer manager's lazy writer.
Usage Notes	The lazy writer is a system process that flushes out batches of dirty, aged buffers and makes them available to user processes. This counter records the number of times per second that SQL Server relocates dirty pages from the buffer pool (in memory) to disk.
Performance	Disk I/O is expensive and you should attempt to provide SQL Server with enough space for the buffer pool that lazy writes are as close to zero as possible. A clear indicator of low memory on the server.
Threshold	Problematic if not 0. If more than 20 you need to increase buffer pool.
Related Measurements	N/A

Disk-related Counters

Moving data onto or off disk is almost always the most time-consuming and expensive operation SQL Server needs to undertake. SQL Server uses built-in mechanisms to avoid the user having to wait while data is being transferred between memory and disk because any slight delay in this process is likely to impact perceived server performance. There are essentially two mechanisms in SQL Server: a buffer cache with pre-loaded data and a plan cache that is loaded with optimal plans detailing the most efficient way to retrieve data. If there are disk performance problems, it may lead you to review the design and implementation of the storage subsystem.

In addition to the Windows counters explained in “Related Windows Counters” on page 228, use the following counters to monitor disk-related issues:

Full Scans/sec

Official Name	SQLServer:Access Methods\Full Scans/sec.
Counter Type	Interval difference counter (rate/second).
Description	Indicates the number of unrestricted full scans per second.
Usage Notes	While table scans are a fact of life, and sometimes faster than index seeks, generally it is better to have fewer table scans than more. This counter is for an entire server, not just a single database.
Performance	Periodic table scans may be attributed to SQL Server internal jobs. However, random spikes in this counter’s values indicate poor or missing indexes.
Threshold	Warning when 1. Error 2 or more.
Related Measurements	N/A

Page Splits/sec

Official Name	SQLServer:Access Methods\Page Splits/sec.
Counter Type	Interval difference counter (rate/second).
Description	Indicates the number of page splits per second that occur as a result of overflowing index pages.
Usage Notes	Page splits are an I/O intensive operation that occur when there is insufficient space in an 8 KB data page to allow an insert or update operation to complete. Under this circumstance, a new page is added and the original data is shared between the two pages before the insert or update takes place.
Performance	While occasional page splitting is normal, excess page splitting can cause excessive disk I/O and contribute to slow performance. These can be avoided through proper index maintenance and good fill factor selection.
Threshold	Warning when over 100.
Related Measurements	N/A

Tip: SQL Server enables autogrowth by default and performs data- and log-file increase when needed. While this may be convenient, it is recommended to manually adjust the setting on the enterprise systems.

Temp Tables Creation Rate

Official Name	SQLServer:General Statistics\Temp Tables Creation Rate.
Counter Type	Interval difference counter (rate/second).
Description	Indicates the number of temporary tables/table variables created per second.
Usage Notes	SQL servers use **tempdb** as a holding area during join, sort, and calculation operations as well as by the version store. Under workloads that make extensive use of **tempdb**, its responsiveness can directly affect the user experience.
Performance	**Tempdb** is a shared global resource. This means that if one database or application is heavily dependent on it, other databases within the same instance may suffer performance problems which are outside their control.
Threshold	N/A
Related Measurements	N/A

Tip: Size **tempdb** sufficiently to ensure no autogrowth will be required.

Lock-related Counters

Locks are necessary for concurrency. SQL Server handles locks automatically. While locks represent the internal behavior of a specific database or the whole SQL Server, and are not related to operating system resources, they have significant impact on response time. Locks are one of the main reasons for long running transactions causing end-user complaints.

In most of the cases, SQL Server resolves locks automatically. However, there are two problematic types of locks that need to be taken care of if they occur consistently:

- **Blocking lock.** Where one process is blocked from locking a resource because another process has already locked it.
- **Deadlock.** When two processes each hold a lock that the other needs to continue. If left alone, they would wait on each other indefinitely.

Average Wait Time (ms)

Official Name	SQL Server:Locks\Average Wait Time (ms).
Counter Type	Average.
Description	Indicates the average amount of wait time (in milliseconds) for each lock request that resulted in a wait.
Usage Notes	Shows if object locking contributes to slow response times. You can use this counter to measure the average wait time of a variety of locks, including database, extent, key, page, RID, and table.
Performance	Watch this counter over time for each of the lock types, finding average values for each type of lock. Then use these average values as a point of reference.
Threshold	N/A
Related Measurements	N/A

Tip: If you can identify one or more types of locks causing transaction delays, then you should investigate further to see if you can identify what specific transactions are causing the locking. Use HP Diagnostics software to catch problematic statements.