PCP Books

May 26, 2020

User’s and Administrator’s Guide

Note: This site is under construction.

CHAPTER 1

Legal Notice

1.1 LICENSE

1.2 TRADEMARKS AND ATTRIBUTIONS

1.2.1 Introduction to PCP

This chapter provides an introduction to Performance Co-Pilot (PCP), an overview of its individual components, and conceptual information to help you use this software.

Contents

Objectives

Performance Co-Pilot (PCP) provides a range of services that may be used to monitor and manage system performance. These services are distributed and scalable to accommodate the most complex system configurations and performance problems.

PCP Target Usage

PCP is targeted at the performance analyst, benchmarker, capacity planner, developer, database administrator, or system administrator with an interest in overall system performance and a need to quickly isolate and understand performance behavior, resource utilization, activity levels, and bottlenecks in complex systems. Platforms that can benefit from this level of performance analysis include large servers, server clusters, or multiserver sites delivering Database Management Systems (DBMS), compute, Web, file, or video services.

Empowering the PCP User

To deal efficiently with the dynamic behavior of complex systems, performance analysts need to filter out noise from the overwhelming stream of performance data, and focus on exceptional scenarios. Visualization of current and historical performance data, and automated reasoning about performance data, effectively provide this filtering.

From the PCP end user's perspective, PCP presents an integrated suite of tools, user interfaces, and services that support real-time and retrospective performance analysis, with a bias towards eliminating mundane information and focusing attention on the exceptional and extraordinary performance behaviors. When this is done, the user can concentrate on in-depth analysis or target management procedures for those critical system performance problems.

Unification of Performance Metric Domains

At the lowest level, performance metrics are collected and managed in autonomous performance domains such as the operating system kernel, a DBMS, a layered service, or an end-user application. These domains feature a multitude of access control policies, access methods, data semantics, and multiversion support. All this detail is irrelevant to the developer or user of a performance monitoring tool, and is hidden by the PCP infrastructure.

Performance Metrics Domain Agents (PMDAs) within PCP encapsulate the knowledge about, and export performance information from, autonomous performance domains.

Uniform Naming and Access to Performance Metrics

Usability and extensibility of performance management tools mandate a single scheme for naming performance metrics. The set of defined names constitutes a Performance Metrics Name Space (PMNS). Within PCP, the PMNS is adaptive so it can be extended, reshaped, and pruned to meet the needs of particular applications and users.

PCP provides a single interface to name and retrieve values for all performance metrics, independently of their source or location.

PCP Distributed Operation

From a purely pragmatic viewpoint, a single workstation must be able to monitor the concurrent performance of multiple remote hosts. At the same time, a single host may be subject to monitoring from multiple remote workstations.

These requirements suggest a classic client-server architecture, which is exactly what PCP uses to provide concurrent and multiconnected access to performance metrics, independent of their host location.

Dynamic Adaptation to Change

Complex systems are subject to continual changes as network connections fail and are reestablished; nodes are taken out of service and rebooted; hardware is added and removed; and software is upgraded, installed, or removed. Often these changes are asynchronous and remote (perhaps in another geographic region or domain of administrative control).

The distributed nature of the PCP (and the modular fashion in which performance metrics domains can be installed, upgraded, and configured on different hosts) enables PCP to adapt concurrently to changes in the monitored system(s).

Variations in the available performance metrics as a consequence of configuration changes are handled automatically and become visible to all clients as soon as the reconfigured host is rebooted or the responsible agent is restarted.

PCP also detects loss of client-server connections, and most clients support subsequent automated reconnection.

Logging and Retrospective Analysis

A range of tools is provided to support flexible, adaptive logging of performance metrics for archive, playback, remote diagnosis, and capacity planning. PCP archive logs may be accumulated either at the host being monitored, at a monitoring workstation, or both.

A universal replay mechanism, modeled on media controls, supports play, step, rewind, fast forward and variable speed processing of archived performance data. Replay for multiple archives, from multiple hosts, is facilitated by an archive aggregation concept.

Most PCP applications are able to process archive logs and real-time performance data with equal facility. Unification of real-time access and access to the archive logs, in conjunction with the media controls, provides powerful mechanisms for building performance tools and to review both current and historical performance data.

Automated Operational Support

For operational and production environments, PCP provides a framework with scripts to customize in order to automate the execution of ongoing tasks such as these:

- Centralized archive logging for multiple remote hosts
- Archive log rotation, consolidation, and culling
- Web-based publishing of charts showing snapshots of performance activity levels in the recent past
- Flexible alarm monitoring: parameterized rules to address common critical performance scenarios and facilities to customize and refine this monitoring
- Retrospective performance audits covering the recent past; for example, daily or weekly checks for performance regressions or quality of service problems

PCP Extensibility

PCP permits the integration of new performance metrics into the PMNS, the collection infrastructure, and the logging framework. The guiding principle is, “if it is important for monitoring system performance, and you can measure it, you can easily integrate it into the PCP framework.”

For many PCP users, the most important performance metrics are not those already supported, but new performance metrics that characterize the essence of good or bad performance at their site, or within their particular application environment.

One example is an application that measures the round-trip time for a benign “probe” transaction against some mission-critical application.

For application developers, a library is provided to support easy-to-use insertion of trace and monitoring points within an application, and the automatic export of resultant performance data into the PCP framework. Other libraries and tools aid the development of customized and fully featured Performance Metrics Domain Agents (PMDAs).

Extensive source code examples are provided in the distribution, and by using the PCP toolkit and interfaces, these customized measures of performance or quality of service can be easily and seamlessly integrated into the PCP framework.

Metric Coverage

The core PCP modules support export of performance metrics that include kernel instrumentation, hardware instrumentation, process-level resource utilization, database and other system services instrumentation, and activity in the PCP collection infrastructure.

The supplied agents support thousands of distinct performance metrics, many of which can have multiple values, for example, per disk, per CPU, or per process.

Conceptual Foundations

The following sections provide a detailed overview of concepts that underpin Performance Co-Pilot (PCP).

Performance Metrics

Across all of the supported performance metric domains, there are a large number of performance metrics. Each metric has its own structure and semantics. PCP presents a uniform interface to these metrics, independent of the underlying metric data source.

The Performance Metrics Name Space (PMNS) provides a hierarchical classification of human-readable metric names, and a mapping from these external names to internal metric identifiers. See Section 1.2.6, “Performance Metrics Name Space”, for a description of the PMNS.

Performance Metric Instances

When performance metric values are returned to a requesting application, there may be more than one value instance for a particular metric; for example, independent counts for each CPU, process, disk, or local filesystem. Internal instance identifiers correspond one to one with external (human-readable) descriptions of the members of an instance domain.

Transient performance metrics (such as per-process information) cause repeated requests for the same metric to return different numbers of values, or changes in the particular instance identifiers returned. These changes are expected and fully supported by the PCP infrastructure; however, metric instantiation is guaranteed to be valid only at the time of collection.

Current Metric Context

When performance metrics are retrieved, they are delivered in the context of a particular source of metrics, a point in time, and a profile of desired instances. This means that the application making the request has already negotiated to establish the context in which the request should be executed.

A metric source may be the current performance data from a particular host (a live or real-time source), or a set of archive logs of performance data collected by pmlogger at some distant host or at an earlier time (a retrospective or archive source).

By default, the collection time for a performance metric is the current time of day for real-time sources, or current point within an archive source. For archives, the collection time may be reset to an arbitrary time within the bounds of the set of archive logs.

Sources of Performance Metrics and Their Domains

Instrumentation for the purpose of performance monitoring typically consists of counts of activity or events, attribution of resource consumption, and service-time or response-time measures. This instrumentation may exist in one or more of the functional domains as shown in Figure 1.1, “Performance Metric Domains as Autonomous Collections of Data”.

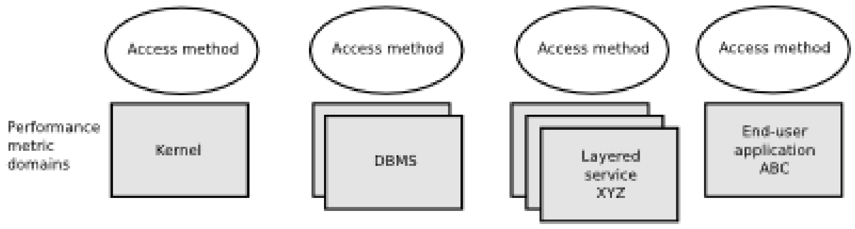

Figure 1.1. Performance Metric Domains as Autonomous Collections of Data

Each domain has an associated access method:

- The operating system kernel, including sub-system data structures - per-process resource consumption, network statistics, disk activity, or memory management instrumentation.
- A layered software service such as activity logs for a World Wide Web server or an email delivery server.
- An application program such as measured response time for a production application running a periodic and benign probe transaction (as often required in service level agreements), or rate of computation and throughput in jobs per minute for a batch stream.
- External equipment such as network routers and bridges.

For each domain, the set of performance metrics may be viewed as an abstract data type, with an associated set of methods that may be used to perform the following tasks:

- Interrogate the metadata that describes the syntax and semantics of the performance metrics
- Control (enable or disable) the collection of some or all of the metrics
- Extract instantiations (current values) for some or all of the metrics

We refer to each functional domain as a performance metrics domain and assume that domains are functionally, architecturally, and administratively independent and autonomous. Obviously the set of performance metrics domains available on any host is variable, and changes with time as software and hardware are installed and removed.

The number of performance metrics domains may be further enlarged in cluster-based or network-based configurations, where there is potentially an instance of each performance metrics domain on each node. Hence, the management of performance metrics domains must be both extensible at a particular host and distributed across a number of hosts.

Each performance metrics domain on a particular host must be assigned a unique Performance Metric Identifier (PMID). In practice, this means unique identifiers are assigned globally for each performance metrics domain type. For example, the same identifier would be used for the Apache Web Server performance metrics domain on all hosts.

Distributed Collection

The performance metrics collection architecture is distributed, in the sense that any performance tool may be executing remotely. However, a PMDA usually runs on the system for which it is collecting performance measurements. In most cases, connecting these tools together on the collector host is the responsibility of the PMCD process, as shown in Figure 1.2, “Process Structure for Distributed Operation”.

Figure 1.2. Process Structure for Distributed Operation

The host running the monitoring tools does not require any collection tools, including pmcd, because all requests for metrics are sent to the pmcd process on the collector host. These requests are then forwarded to the appropriate PMDAs, which respond with metric descriptions, help text, and most importantly, metric values.

The connections between monitor clients and pmcd processes are managed in libpcp, below the PMAPI level; see the pmapi(3) man page. Connections between PMDAs and pmcd are managed by the PMDA routines; see the pmda(3) man page. There can be multiple monitor clients and multiple PMDAs on the one host, but normally there would be only one pmcd process.

Performance Metrics Name Space

Internally, each unique performance metric is identified by a Performance Metric Identifier (PMID) drawn from a universal set of identifiers, including some that are reserved for site-specific, application-specific, and customer-specific use.

An external name space - the Performance Metrics Name Space (PMNS) - maps from a hierarchy (or tree) of human-readable names to PMIDs.

Performance Metrics Name Space Diagram

Each node in the PMNS tree is assigned a label that must begin with an alphabet character, and be followed by zero or more alphanumeric characters or the underscore (_) character. The root node of the tree has the special label of root.

A metric name is formed by traversing the tree from the root to a leaf node with each node label on the path separated by a period. The common prefix root. is omitted from all names. For example, Figure 1.3, "Small Performance Metrics Name Space (PMNS) " shows the nodes in a small subsection of a PMNS.

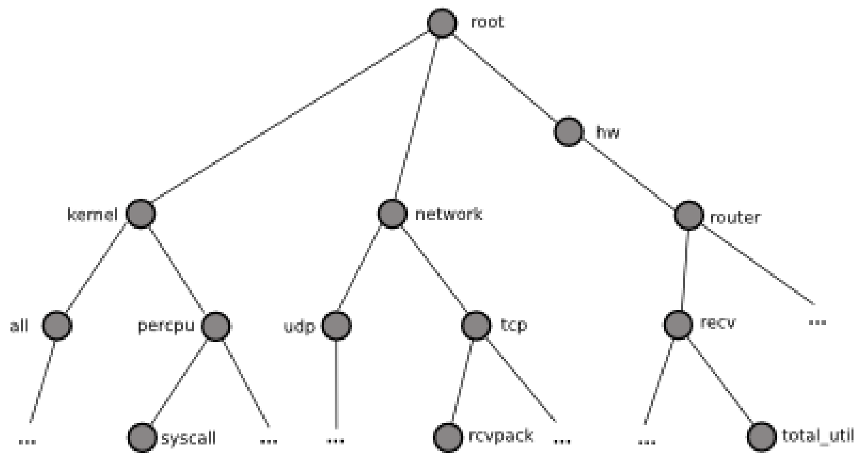

Figure 1.3. Small Performance Metrics Name Space (PMNS)

In this subsection, the following are valid names for performance metrics:

```
kernel.percpu.syscall
network.tcp.rcvpack
hw.router.recv.total_util
```

Descriptions for Performance Metrics

Through the various performance metric domains, the PCP must support a wide range of formats and semantics for performance metrics. This metadata describing the performance metrics includes the following:

- The internal identifier, Performance Metric Identifier (PMID), for the metric
- The format and encoding for the values of the metric, for example, an unsigned 32-bit integer or a string or a 64-bit IEEE format floating point number
- The semantics of the metric, particularly the interpretation of the values as free-running counters or instantaneous values
- The dimensionality of the values, in the dimensions of events, space, and time
- The scale of values; for example, bytes, kilobytes (KB), or megabytes (MB) for the space dimension
- An indication if the metric may have one or many associated values
- Short (and extended) help text describing the metric

For each metric, this metadata is defined within the associated PMDA, and PCP arranges for the information to be exported to performance tools that use the metadata when interpreting the values for each metric.

Values for Performance Metrics

The following sections describe two types of performance metrics, single-valued and set-valued.

Single-Valued Performance Metrics

Some performance metrics have a singular value within their performance metric domains. For example, available memory (or the total number of context switches) has only one value per performance metric domain, that is, one value per host. The metadata describing the metric makes this fact known to applications that process values for these metrics.

Set-Valued Performance Metrics

Some performance metrics have a set of values or instances in each implementing performance metric domain. For example, one value for each disk, one value for each process, one value for each CPU, or one value for each activation of a given application.

When a metric has multiple instances, the PCP framework does not pollute the Name Space with additional metric names; rather, a single metric may have an associated set of values. These multiple values are associated with the members of an instance domain, such that each instance has a unique instance identifier within the associated instance domain. For example, the “per CPU” instance domain may use the instance identifiers 0, 1, 2, 3, and so on to identify the configured processors in the system.

Internally, instance identifiers are encoded as binary values, but each performance metric domain also supports corresponding strings as external names for the instance identifiers, and these names are used at the user interface to the PCP utilities.

For example, the performance metric disk.dev.total counts I/O operations for each disk spindle, and the associated instance domain contains one member for each disk spindle. On a system with five specific disks, one value would be associated with each of the external and internal instance identifier pairs shown in Table 1.1, “Sample Instance Identifiers for Disk Statistics ”.

Table 1.1. Sample Instance Identifiers for Disk Statistics

External Instance Identifier	Internal Instance Identifier
disk0	131329
disk1	131330
disk2	131331
disk3	131841
disk4	131842

Multiple performance metrics may be associated with a single instance domain.

Each performance metric domain may dynamically establish the instances within an instance domain. For example, there may be one instance for the metric kernel.percpu.idle on a workstation, but multiple instances on a multiprocessor server. Even more dynamic is filesys.free, where the values report the amount of free space per file system, and the number of values tracks the mounting and unmounting of local filesystems.

PCP arranges for information describing instance domains to be exported from the performance metric domains to the applications that require this information. Applications may also choose to retrieve values for all instances of a performance metric, or some arbitrary subset of the available instances.

Collector and Monitor Roles

Hosts supporting PCP services are broadly classified into two categories:

Collector : Hosts that have pmcd and one or more performance metric domain agents (PMDAs) running to collect and export performance metrics

Monitor : Hosts that import performance metrics from one or more collector hosts to be consumed by tools to monitor, manage, or record the performance of the collector hosts

Each PCP enabled host can operate as a collector, a monitor, or both.

Retrospective Sources of Performance Metrics

The PMAPI also supports delivery of performance metrics from a historical source in the form of a PCP archive log. Archive logs are created using the pmlogger utility, and are replayed in an architecture as shown in Figure 1.4, “Architecture for Retrospective Analysis”.

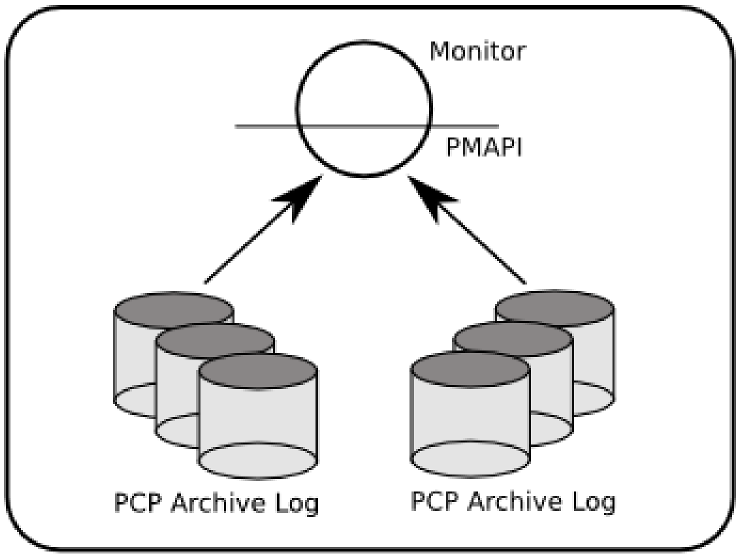

Figure 1.4. Architecture for Retrospective Analysis

The PMAPI has been designed to minimize the differences required for an application to process performance data from an archive or from a real-time source. As a result, most PCP tools support live and retrospective monitoring with equal facility.

Product Extensibility

Much of the PCP software's potential for attacking difficult performance problems in production environments comes from the design philosophy that considers extensibility to be critically important.

The performance analyst can take advantage of the PCP infrastructure to deploy value-added performance monitoring tools and services. Here are some examples:

- Easy extension of the PCP collector to accommodate new performance metrics and new sources of performance metrics, in particular using the interfaces of a special-purpose library to develop new PMDAs (see the pmda(3) man page)
- Use of libraries (libpcp_pmda and libpcp_mmv) to aid in the development of new capabilities to export performance metrics from local applications
- Operation on any performance metric using generalized toolkits
- Distribution of PCP components such as collectors across the network, placing the service where it can do the most good
- Dynamic adjustment to changes in system configuration
- Flexible customization built into the design of all PCP tools
- Creation of new monitor applications, using the routines described in the pmapi(3) man page

Overview of Component Software

Performance Co-Pilot (PCP) is composed of both text-based and graphical tools. Each tool is fully documented by a man page. These man pages are named after the tools or commands they describe, and are accessible through the man command. For example, to see the pminfo(1) man page for the pminfo command, enter this command:

```
man pminfo
```

A representative list of PCP tools and commands, grouped by functionality, is provided in the following four sections.

Performance Monitoring and Visualization

The following tools provide the principal services for the PCP end-user with an interest in monitoring, visualizing, or processing performance information collected either in real time or from PCP archive logs:

pcp-atop

Full-screen monitor of the load on a system from a kernel, hardware and processes point of view. It is modeled on the Linux atop(1) tool (home page) and provides a showcase for the variety of data available using PCP services and the Python scripting interfaces.

pmchart

Strip chart tool for arbitrary performance metrics. Interactive graphical utility that can display multiple charts simultaneously, from multiple hosts or set of archives, aligned on a unified time axis (X-axis), or on multiple tabs.

pcp-collectl

Statistics collection tool with good coverage of a number of Linux kernel subsystems, with the everything-in-one-tool approach pioneered by sar(1). It is modeled on the Linux collectl(1) utility (home page) and provides another example of use of the Python scripting interfaces to build more complex functionality with relative ease, with PCP as a foundation.

pmrep

Outputs the values of arbitrary performance metrics collected live or from a single PCP archive, in textual format.

pmevent

Reports on event metrics, decoding the timestamp and event parameters for text-based reporting.

pmie

Evaluates predicate-action rules over performance metrics for alarms, automated system management tasks, dynamic configuration tuning, and so on. It is an inference engine.

pmieconf

Creates parameterized rules to be used with the PCP inference engine (pmie). It can be run either interactively or from scripts for automating the setup of inference (the PCP start scripts do this, for example, to generate a default configuration).

pminfo

Displays information about arbitrary performance metrics available from PCP, including help text with -T.

pmlogsummary

Calculates and reports various statistical summaries of the performance metric values from a set of PCP archives.

pmprobe

Probes for performance metric availability, values, and instances.

pmstat

Provides a text-based display of metrics that summarize the performance of one or more systems at a high level.

pmval

Provides a text-based display of the values for arbitrary instances of a selected performance metric, suitable for ASCII logs or inquiry over a slow link.

Collecting, Transporting, and Archiving Performance Information

PCP provides the following tools to support real-time data collection, network transport, and archive log creation services for performance data:

mkaf

Aggregates an arbitrary collection of PCP archive logs into a folio to be used with pmafm.

pmafm

Interrogates, manages, and replays an archive folio as created by mkaf, or the periodic archive log management scripts, or the record mode of other PCP tools.

pmcd

Is the Performance Metrics Collection Daemon (PMCD). This daemon must run on each system being monitored, to collect and export the performance information necessary to monitor the system.

pmcd_wait

Waits for pmcd to be ready to accept client connections.

pmdaapache

Exports performance metrics from the Apache Web Server. It is a Performance Metrics Domain Agent (PMDA).

pmdacisco

Extracts performance metrics from one or more Cisco routers.

pmdaelasticseach

Extracts performance metrics from an elasticsearch cluster.

pmdagfs2

Exports performance metrics from the GFS2 clustered filesystem.

pmdagluster

Extracts performance metrics from the Gluster filesystem.

pmdainfiniband

Exports performance metrics from the Infiniband kernel driver.

pmdakvm

Extracts performance metrics from the Linux Kernel Virtual Machine (KVM) infrastructure.

pmdalustrecomm

Exports performance metrics from the Lustre clustered filesystem.

pmdamailq

Exports performance metrics describing the current state of items in the sendmail queue.

pmdamemcache

Extracts performance metrics from memcached, a distributed memory caching daemon commonly used to improve web serving performance.

pmdammv

Exports metrics from instrumented applications linked with the pcp_mmv shared library or the Parfait framework for Java instrumentation. These metrics are custom developed per application, and in the case of Parfait, automatically include numerous JVM, Tomcat and other server or container statistics.

pmdamysql

Extracts performance metrics from the MySQL relational database.

pmdanamed

Exports performance metrics from the Internet domain name server, named.

pmdanginx

Extracts performance metrics from the nginx HTTP and reverse proxy server.

pmdapostfix

Export performance metrics from the Postfix mail transfer agent.

pmdapostgres

Extracts performance metrics from the PostgreSQL relational database.

pmdaproc

Exports performance metrics for running processes.

pmdarsyslog

Extracts performance metrics from the Reliable System Log daemon.

pmdasamba

Extracts performance metrics from Samba, a Windows SMB/CIFS server.

pmdasendmail

Exports mail activity statistics from sendmail.

pmdashping

Exports performance metrics for the availability and quality of service (response-time) for arbitrary shell commands.

pmdasnmp

Extracts SNMP performance metrics from local or remote SNMP-enabled devices.

pmdasummary

Derives performance metrics values from values made available by other PMDAs. It is a PMDA itself.

pmdasystemd

Extracts performance metrics from the systemd and journald services.

pmdatrace

Exports transaction performance metrics from application processes that use the pcp_trace library.

pmdavmware

Extracts performance metrics from a VMWare virtualization host.

pmdaweblog

Scans Web-server logs to extract metrics characterizing.

pmdaxfs

Extracts performance metrics from the Linux kernel XFS filesystem implementation.

pmdumplog

Displays selected state information, control data, and metric values from a set of PCP archive logs created by pmlogger.

pmlc

Exercises control over an instance of the PCP archive logger pmlogger, to modify the profile of which metrics are logged and/or how frequently their values are logged.

pmlogcheck

Performs integrity check for individual PCP archives.

pmlogconf

Creates or modifies pmlogger configuration files for many common logging scenarios, optionally probing for available metrics and enabled functionality. It can be run either interactively or from scripts for automating the setup of data logging (the PCP start scripts do this, for example, to generate a default configuration).

pmlogextract

Reads one or more PCP archive logs and creates a temporally merged and reduced PCP archive log as output.

pmlogger

Creates PCP archive logs of performance metrics over time. Many tools accept these PCP archive logs as alternative sources of metrics for retrospective analysis.

pmproxy

Provides REST APIs, archive discovery, and both PCP and Redis protocol proxying when executing PCP or Redis client tools through a network firewall system.

pmtrace

Provides a simple command line interface to the trace PMDA and its associated pcp_trace library.

Operational and Infrastructure Support

PCP provides the following tools to support the PCP infrastructure and assist operational procedures for PCP deployment in a production environment:

pcp

Summarizes that state of a PCP installation.

pmdbg

Describes the available facilities and associated control flags. PCP tools include internal diagnostic and debugging facilities that may be activated by run-time flags.

pmerr

Translates PCP error codes into human-readable error messages.

pmhostname

Reports hostname as returned by gethostbyname. Used in assorted PCP management scripts.

pmie_check

Administration of the Performance Co-Pilot inference engine (pmie).

pmlock

Attempts to acquire an exclusive lock by creating a file with a mode of 0.

pmlogger_

Allows you to create a customized regime of administration and management for PCP archive log files. The **pmlogger_check**, **pmlogger_daily**, and **pmlogger_merge** scripts are intended for periodic execution via the cron command.

pmnewlog

Performs archive log rotation by stopping and restarting an instance of pmlogger.

pmnsadd

Adds a subtree of new names into a PMNS, as used by the components of PCP.

pmnsdel

Removes a subtree of names from a PMNS, as used by the components of the PCP.

pmnsmerge

Merges multiple PMNS files together, as used by the components of PCP.

pmstore

Reinitializes counters or assigns new values to metrics that act as control variables. The command changes the current values for the specified instances of a single performance metric.

Application and Agent Development

The following PCP tools aid the development of new programs to consume performance data, and new agents to export performance data within the PCP framework:

chkhelp

Checks the consistency of performance metrics help database files.

dbpmda

Allows PMDA behavior to be exercised and tested. It is an interactive debugger for PMDAs.

newhelp

Generates the database files for one or more source files of PCP help text.

pmapi

Defines a procedural interface for developing PCP client applications. It is the Performance Metrics Application Programming Interface (PMAPI).

pmclient

Is a simple client that uses the PMAPI to report some high-level system performance metrics.

pmda

Is a library used by many shipped PMDAs to communicate with a pmcd process. It can expedite the development of new and custom PMDAs.

pmgenmap

Generates C declarations and cpp(1) macros to aid the development of customized programs that use the facilities of PCP. It is a PMDA development tool.

1.2.2 Installing and Configuring Performance Co-Pilot

The sections in this chapter describe the basic installation and configuration steps necessary to run Performance Co-Pilot (PCP) on your systems.

Contents

Product Structure

In a typical deployment, Performance Co-Pilot (PCP) would be installed in a collector configuration on one or more hosts, from which the performance information could then be collected, and in a monitor configuration on one or more workstations, from which the performance of the server systems could then be monitored.

On some platforms Performance Co-Pilot is presented as multiple packages; typically separating the server components from graphical user interfaces and documentation.

pcp-X.Y.Z-rev

package for core PCP

pcp-gui-X.Y.Z-rev

package for graphical PCP client tools

pcp-doc-X.Y.Z-rev

package for online PCP documentation

Performance Metrics Collection Daemon (PMCD)

On each Performance Co-Pilot (PCP) collection system, you must be certain that the `pmcd` daemon is running. This daemon coordinates the gathering and exporting of performance statistics in response to requests from the PCP monitoring tools.

Starting and Stopping the PMCD

To start the daemon, enter the following commands as `root` on each PCP collection system:

```
chkconfig pmcd on
${PCP_RC_DIR}/pmcd start
```

These commands instruct the system to start the daemon immediately, and again whenever the system is booted. It is not necessary to start the daemon on the monitoring system unless you wish to collect performance information from it as well.

To stop pmcd immediately on a PCP collection system, enter the following command:

```
${PCP_RC_DIR}/pmcd stop
```

Restarting an Unresponsive PMCD

Sometimes, if a daemon is not responding on a PCP collection system, the problem can be resolved by stopping and then immediately restarting a fresh instance of the daemon. If you need to stop and then immediately restart PMCD on a PCP collection system, use the `start` argument provided with the script in `${PCP_RC_DIR}`. The command syntax is, as follows:

```
${PCP_RC_DIR}/pmcd start
```

On startup, `pmcd` looks for a configuration file at `${PCP_PMCDCONF_PATH}`. This file specifies which agents cover which performance metrics domains and how PMCD should make contact with the agents. A comprehensive description of the configuration file syntax and semantics can be found in the `pmcd(1)` man page.

If the configuration is changed, `pmcd` reconfigures itself when it receives the `SIGHUP` signal. Use the following command to send the `SIGHUP` signal to the daemon:

```
${PCP_BINADM_DIR}/pmsignal -a -s HUP pmcd
```

This is also useful when one of the PMDAs managed by `pmcd` has failed or has been terminated by `pmcd`. Upon receipt of the `SIGHUP` signal, `pmcd` restarts any PMDA that is configured but inactive. The exception to this rule is the case of a PMDA which must run with superuser privileges (where possible, this is avoided) - for these PMDAs, a full `pmcd` restart must be performed, using the process described earlier (not SIGHUP).

PMCD Diagnostics and Error Messages

If there is a problem with `pmcd`, the first place to investigate should be the `pmcd.log` file. By default, this file is in the `${PCP_LOG_DIR}/pmcd` directory.

PMCD Options and Configuration Files

There are two files that control PMCD operation. These are the `${PCP_PMCDCONF_PATH}` and `${PCP_PMCDOPTIONS_PATH}` files. The `pmcd.options` file contains the command line options used with PMCD; it is read when the daemon is invoked by `${PCP_RC_DIR}/pmcd`. The `pmcd.conf` file contains configuration information regarding domain agents and the metrics that they monitor. These configuration files are described in the following sections.

The pmcd.options File

Command line options for the PMCD are stored in the `${PCP_PMCDOPTIONS_PATH}` file. The PMCD can be invoked directly from a shell prompt, or it can be invoked by `${PCP_RC_DIR}/pmcd` as part of the boot process. It is usual and normal to invoke it using `${PCP_RC_DIR}/pmcd`, reserving shell invocation for debugging purposes.

The PMCD accepts certain command line options to control its execution, and these options are placed in the `pmcd.options` file when `${PCP_RC_DIR}/pmcd` is being used to start the daemon. The following options (amongst others) are available:

-i *address*	For hosts with more than one network interface, this option specifies the interface on which this instance of the PMCD accepts connections. Multiple `-i` options may be specified. The default in the absence of any `-i` option is for PMCD to accept connections on all interfaces.
-l *file*	Specifies a log file. If no `-l` option is specified, the log file name is `pmcd.log` and it is created in the directory `${PCP_LOG_DIR}/pmcd/`.
-s *file*	Specifies the path to a local unix domain socket (for platforms supporting this socket family only). The default value is `${PCP_RUN_DIR}/pmcd.socket`.
-t *seconds*	Specifies the amount of time, in seconds, before PMCD times out on protocol data unit (PDU) exchanges with PMDAs. If no time out is specified, the default is five seconds. Setting time out to zero disables time outs (not recommended, PMDAs should always respond quickly).The time out may be dynamically modified by storing the number of seconds into the metric `pmcd.control.timeout` using `pmstore`.
-T *mask*	Specifies whether connection and PDU tracing are turned on for debugging purposes.

See the `pmcd(1)` man page for complete information on these options.

The default `pmcd.options` file shipped with PCP is similar to the following:

```
# command-line options to pmcd, uncomment/edit lines as required

# longer timeout delay for slow agents
# -t 10

# suppress timeouts
# -t 0

# make log go someplace else
# -l /some/place/else

# debugging knobs, see pmdbg(1)
# -D N
# -f

# Restricting (further) incoming PDU size to prevent DOS attacks
# -L 16384

# enable event tracing bit fields
#   1   trace connections
#   2   trace PDUs
# 256   unbuffered tracing
# -T 3

# setting of environment variables for pmcd and
# the PCP rc scripts. See pmcd(1) and PMAPI(3).
# PMCD_WAIT_TIMEOUT=120
```

The most commonly used options have been placed in this file for your convenience. To uncomment and use an option, simply remove the pound sign (#) at the beginning of the line with the option you wish to use. Restart `pmcd` for the change to take effect; that is, as superuser, enter the command:

```
${PCP_RC_DIR}/pmcd start
```

The pmcd.conf File

When the PMCD is invoked, it reads its configuration file, which is `${PCP_PMCDCONF_PATH}`. This file contains entries that specify the PMDAs used by this instance of the PMCD and which metrics are covered by these PMDAs. Also, you may specify access control rules in this file for the various hosts, users and groups on your network. This file is described completely in the pmcd(1) man page.

With standard PCP operation (even if you have not created and added your own PMDAs), you might need to edit this file in order to add any additional access control you wish to impose. If you do not add access control rules, all access for all operations is granted to the local host, and read-only access is granted to remote hosts. The `pmcd.conf` file is automatically generated during the software build process and on Linux, for example, is similar to the following:

```
 Performance Metrics Domain Specifications
#
# This file is automatically generated during the build
# Name  Id      IPC     IPC Params      File/Cmd
root    1       pipe    binary          /var/lib/pcp/pmdas/root/pmdaroot
pmcd    2       dso     pmcd_init       ${PCP_PMDAS_DIR}/pmcd/pmda_pmcd.so
proc    3       pipe    binary          ${PCP_PMDAS_DIR}/proc/pmdaproc -d 3
xfs     11      pipe    binary          ${PCP_PMDAS_DIR}/xfs/pmdaxfs -d 11
linux   60      dso     linux_init      ${PCP_PMDAS_DIR}/linux/pmda_linux.so
mmv     70      dso     mmv_init        /var/lib/pcp/pmdas/mmv/pmda_mmv.so

[access]
disallow ".*" : store;
disallow ":*" : store;
allow "local:*" : all;
```

Note: Even though PMCD does not run with root privileges, you must be very careful not to configure PMDAs in this file if you are not sure of their action. This is because all PMDAs are initially started as root (allowing them to assume alternate identities, such as postgres for example), after which pmcd drops its privileges. Pay close attention that permissions on this file are not inadvertently downgraded to allow public write access.

Each entry in this configuration file contains rules that specify how to connect the PMCD to a particular PMDA and which metrics the PMDA monitors. A PMDA may be attached as a Dynamic Shared Object (DSO) or by using a socket or a pair of pipes. The distinction between these attachment methods is described below.

An entry in the `pmcd.conf` file looks like this:

```
label_name   domain_number   type   path
```

The *label_name* field specifies a name for the PMDA. The *domain_number* is an integer value that specifies a domain of metrics for the PMDA. The *type* field indicates the type of entry (DSO, socket, or pipe). The *path* field is for additional information, and varies according to the type of entry.

The following rules are common to DSO, socket, and pipe syntax:

label_name

An alphanumeric string identifying the agent.

domain_number

An unsigned integer specifying the agent's domain.

DSO entries follow this syntax:

```
label_name domain_number dso entry-point path
```

The following rules apply to the DSO syntax:

dso

The entry type.

entry-point

The name of an initialization function called when the DSO is loaded.

path

Designates the location of the DSO. An absolute path must be used. On most platforms this will be a `so` suffixed file, on Windows it is a `dll`, and on Mac OS X it is a `dylib` file.

Socket entries in the `pmcd.conf` file follow this syntax:

```
label_name domain_number socket addr_family address command [args]
```

The following rules apply to the socket syntax:

socket

The entry type.

addr_family

Specifies if the socket is AF_INET, AF_IPV6 or AF_UNIX. If the socket is INET, the word inet appears in this place. If the socket is IPV6, the word ipv6 appears in this place. If the socket is UNIX, the word unix appears in this place.

address

Specifies the address of the socket. For INET or IPv6 sockets, this is a port number or port name. For UNIX sockets, this is the name of the PMDA's socket on the local host.

command

Specifies a command to start the PMDA when the PMCD is invoked and reads the configuration file.

args

Optional arguments for command.

Pipe entries in the `pmcd.conf` file follow this syntax:

```
label_name domain_number pipe protocol command [args]
```

The following rules apply to the pipe syntax:

pipe

The entry type.

protocol

Specifies whether a text-based or a binary PCP protocol should be used over the pipes. Historically, this parameter was able to be "text" or "binary." The text-based protocol has long since been deprecated and removed, however, so nowadays "binary" is the only valid value here.

command

Specifies a command to start the PMDA when the PMCD is invoked and reads the configuration file.

args

Optional arguments for command.

Controlling Access to PMCD with pmcd.conf

You can place this option extension in the `pmcd.conf` file to control access to performance metric data based on hosts, users and groups. To add an access control section, begin by placing the following line at the end of your `pmcd.conf` file:

```
[access]
```

Below this line, you can add entries of the following forms:

```
allow hosts hostlist : operations ;   disallow hosts hostlist : operations ;
allow users userlist : operations ;   disallow users userlist : operations ;
allow groups grouplist : operations ;   disallow groups grouplist : operations ;
```

The keywords users, groups and hosts can be used in either plural or singular form.

The userlist and grouplist fields are comma-separated lists of authenticated users and groups from the local `/etc/passwd` and `/etc/groups` files, NIS (network information service) or LDAP (lightweight directory access protocol) service.

The hostlist is a comma-separated list of host identifiers; the following rules apply:

Host names must be in the local system's `/etc/hosts` file or known to the local DNS (domain name service).

IP and IPv6 addresses may be given in the usual numeric notations.

A wildcarded IP or IPv6 address may be used to specify groups of hosts, with the single wildcard character * as the last-given component of the address. The wildcard .* refers to all IP (IPv4) addresses. The wildcard :* refers to all IPv6 addresses. If an IPv6 wildcard contains a :: component, then the final * refers to the final 16 bits of the address only, otherwise it refers to the remaining unspecified bits of the address.

The wildcard ''*" refers to all users, groups or host addresses. Names of users, groups or hosts may not be wildcarded.

For example, the following hostlist entries are all valid:

```
babylon
babylon.acme.com
123.101.27.44
localhost
155.116.24.*
192.*
.*
fe80::223:14ff:feaf:b62c
fe80::223:14ff:feaf:*
fe80:*
:*
*
```

The operations field can be any of the following:

A comma-separated list of the operation types described below.

The word all to allow or disallow all operations as specified in the first field.

The words all except and a list of operations. This entry allows or disallows all operations as specified in the first field except those listed.

The phrase maximum N connections to set an upper bound (N) on the number of connections an individual host, user or group of users may make. This can only be added to the operations list of an allow statement.

The operations that can be allowed or disallowed are as follows:

fetch

Allows retrieval of information from the PMCD. This may be information about a metric (such as a description, instance domain, or help text) or an actual value for a metric.

store

Allows the PMCD to store metric values in PMDAs that permit store operations. Be cautious in allowing this operation, because it may be a security opening in large networks, although the PMDAs shipped with the PCP package typically reject store operations, except for selected performance metrics where the effect is benign.

For example, here is a sample access control portion of a `${PCP_PMCDCONF_PATH}` file:

```
allow hosts babylon, moomba : all ;
disallow user sam : all ;
allow group dev : fetch ;
allow hosts 192.127.4.* : fetch ;
disallow host gate-inet : store ;
```

Complete information on access control syntax rules in the `pmcd.conf` file can be found in the `pmcd(1)` man page.

Managing Optional PMDAs

Some Performance Metrics Domain Agents (PMDAs) shipped with Performance Co-Pilot (PCP) are designed to be installed and activated on every collector host, for example, linux, windows, darwin, pmcd, and process PMDAs.

Other PMDAs are designed for optional activation and require some user action to make them operational. In some cases these PMDAs expect local site customization to reflect the operational environment, the system configuration, or the production workload. This customization is typically supported by interactive installation scripts for each PMDA.

Each PMDA has its own directory located below `${PCP_PMDAS_DIR}`. Each directory contains a Remove script to unconfigure the PMDA, remove the associated metrics from the PMNS, and restart the pmcd daemon; and an Install script to install the PMDA, update the PMNS, and restart the PMCD daemon.

As a shortcut mechanism to support automated PMDA installation, a file named .NeedInstall can be created in a PMDA directory below `${PCP_PMDAS_DIR}`. The next restart of PCP services will invoke that PMDAs installation automatically, with default options taken.

PMDA Installation on a PCP Collector Host

To install a PMDA you must perform a collector installation for each host on which the PMDA is required to export performance metrics. PCP provides a distributed metric namespace (PMNS) and metadata, so it is not necessary to install PMDAs (with their associated PMNS) on PCP monitor hosts.

You need to update the PMNS, configure the PMDA, and notify PMCD. The Install script for each PMDA automates these operations, as follows:

1. Log in as root (the superuser).
2. Change to the PMDA's directory as shown in the following example:

   ```
   cd ${PCP_PMDAS_DIR}/cisco
   ```

3. In the unlikely event that you wish to use a non-default Performance Metrics Domain (PMD) assignment, determine the current PMD assignment:

   ```
   cat domain.h
   ```

Check that there is no conflict in the PMDs as defined in `${PCP_VAR_DIR}/pmns/stdpmid` and the other PMDAs currently in use (listed in `${PCP_PMCDCONF_PATH}`). Edit `domain.h` to assign the new domain number if there is a conflict (this is highly unlikely to occur in a regular PCP installation).

4. Enter the following command:

   ```
   ./Install
   ```

You may be prompted to enter some local parameters or configuration options. The script applies all required changes to the control files and to the PMNS, and then notifies PMCD. Example 2.1, “PMNS Installation Output ” is illustrative of the interactions:

Example 2.1. PMNS Installation Output:

```
Cisco hostname or IP address? [return to quit] wanmelb

A user-level password may be required for Cisco “show int” command.
    If you are unsure, try the command
        $ telnet wanmelb
    and if the prompt “Password:” appears, a user-level password is
    required; otherwise answer the next question with an empty line.

User-level Cisco password? ********
Probing Cisco for list of interfaces ...

Enter interfaces to monitor, one per line in the format
tX where “t” is a type and one of “e” (Ethernet), or “f” (Fddi), or
“s” (Serial), or “a” (ATM), and “X” is an interface identifier
which is either an integer (e.g.  4000 Series routers) or two
integers separated by a slash (e.g. 7000 Series routers).

The currently unselected interfaces for the Cisco “wanmelb” are:
    e0 s0 s1
Enter “quit” to terminate the interface selection process.
Interface? [e0] s0

The currently unselected interfaces for the Cisco “wanmelb” are:
       e0 s1
Enter “quit” to terminate the interface selection process.
Interface? [e0] s1

The currently unselected interfaces for the Cisco “wanmelb” are:
    e0
Enter “quit” to terminate the interface selection process.
Interface? [e0] quit

Cisco hostname or IP address? [return to quit]
Updating the Performance Metrics Name Space (PMNS) ...
Installing pmchart view(s) ...
Terminate PMDA if already installed ...
Installing files ...
Updating the PMCD control file, and notifying PMCD ...
Check cisco metrics have appeared ... 5 metrics and 10 values
```

PMDA Removal on a PCP Collector Host

To remove a PMDA, you must perform a collector removal for each host on which the PMDA is currently installed.

The PMNS needs to be updated, the PMDA unconfigured, and PMCD notified. The Remove script for each PMDA automates these operations, as follows:

1. Log in as root (the superuser).
2. Change to the PMDA's directory as shown in the following example:

   ```
   cd ${PCP_PMDAS_DIR}/elasticsearch
   ```

3. Enter the following command:

   ```
   ./Remove
   ```

The following output illustrates the result:

```
Culling the Performance Metrics Name Space ...
elasticsearch ... done
Updating the PMCD control file, and notifying PMCD ...
Removing files ...
Check elasticsearch metrics have gone away ... OK
```

Troubleshooting

The following sections offer troubleshooting advice on the Performance Metrics Name Space (PMNS), missing and incomplete values for performance metrics, kernel metrics and the PMCD.

Advice for troubleshooting the archive logging system is provided in Chapter 6, Archive Logging.

Performance Metrics Name Space

To display the active PMNS, use the `pminfo` command; see the `pminfo(1)` man page.

The PMNS at the collector host is updated whenever a PMDA is installed or removed, and may also be updated when new versions of PCP are installed. During these operations, the PMNS is typically updated by merging the (plaintext) namespace components from each installed PMDA. These separate PMNS components reside in the `${PCP_VAR_DIR}/pmns` directory and are merged into the `root` file there.

Missing and Incomplete Values for Performance Metrics

Missing or incomplete performance metric values are the result of their unavailability.

Metric Values Not Available

The following symptom has a known cause and resolution:

Symptom:

Values for some or all of the instances of a performance metric are not available.

Cause:

This can occur as a consequence of changes in the installation of modules (for example, a DBMS or an application package) that provide the performance instrumentation underpinning the PMDAs. Changes in the selection of modules that are installed or operational, along with changes in the version of these modules, may make metrics appear and disappear over time.

In simple terms, the PMNS contains a metric name, but when that metric is requested, no PMDA at the collector host supports the metric.

For archive logs, the collection of metrics to be logged is a subset of the metrics available, so utilities replaying from a PCP archive log may not have access to all of the metrics available from a live (PMCD) source.

Resolution:

Make sure the underlying instrumentation is available and the module is active. Ensure that the PMDA is running on the host to be monitored. If necessary, create a new archive log with a wider range of metrics to be logged.

Kernel Metrics and the PMCD

The following issues involve the kernel metrics and the PMCD:

- Cannot connect to remote PMCD
- PMCD not reconfiguring after hang-up
- PMCD does not start

Cannot Connect to Remote PMCD

The following symptom has a known cause and resolution:

Symptom:

A PCP client tool (such as pmchart, pmie, or pmlogger) complains that it is unable to connect to a remote PMCD (or establish a PMAPI context), but you are sure that PMCD is active on the remote host.

Cause:

To avoid hanging applications for the duration of TCP/IP time outs, the PMAPI library implements its own time out when trying to establish a connection to a PMCD. If the connection to the host is over a slow network, then successful establishment of the connection may not be possible before the time out, and the attempt is abandoned.

Alternatively, there may be a firewall in-between the client tool and PMCD which is blocking the connection attempt.

Finally, PMCD may be running in a mode where it does not acept remote connections, or only listening on certain interface.

Resolution:

Establish that the PMCD on far-away-host is really alive, by connecting to its control port (TCP port number 44321 by default):

```
telnet far-away-host 44321
```

This response indicates the PMCD is not running and needs restarting:

```
Unable to connect to remote host: Connection refused
```

To restart the PMCD on that host, enter the following command:

```
${PCP_RC_DIR}/pmcd start
```

This response indicates the PMCD is running:

```
Connected to far-away-host
```

Interrupt the telnet session, increase the `PMAPI` time out by setting the `PMCD_CONNECT_TIMEOUT` environment variable to some number of seconds (60 for instance), and try the PCP client tool again.

Verify that PMCD is not running in local-only mode, by looking for an enabled value (one) from:

```
pminfo -f pmcd.feature.local
```

This setting is controlled from the `PMCD_LOCAL` environment variable usually set via `${PCP_SYSCONFIG_DIR}/pmcd`.

If these techniques are ineffective, it is likely an intermediary firewall is blocking the client from accessing the PMCD port - resolving such issues is firewall-host platform-specific and cannot practically be covered here.

PMCD Not Reconfiguring after SIGHUP

The following symptom has a known cause and resolution:

Symptom

PMCD does not reconfigure itself after receiving the SIGHUP signal.

Cause

If there is a syntax error in `${PCP_PMCDCONF_PATH}`, PMCD does not use the contents of the file. This can lead to situations in which the configuration file and PMCD's internal state do not agree.

Resolution

Always monitor PMCD's log. For example, use the following command in another window when reconfiguring PMCD, to watch errors occur:

```
tail -f ${PCP_LOG_DIR}/pmcd/pmcd.log
```

PMCD Does Not Start

The following symptom has a known cause and resolution:

Symptom

If the following messages appear in the PMCD log (`${PCP_LOG_DIR}/pmcd/pmcd.log`), consider the cause and resolution:

```
pcp[27020] Error: OpenRequestSocket(44321) bind: Address already in
use
pcp[27020] Error: pmcd is already running
pcp[27020] Error: pmcd not started due to errors!
```

Cause

PMCD is already running or was terminated before it could clean up properly. The error occurs because the socket it advertises for client connections is already being used or has not been cleared by the kernel.

Resolution

Start PMCD as root (superuser) by typing:

```
${PCP_RC_DIR}/pmcd start
```

Any existing PMCD is shut down, and a new one is started in such a way that the symptomatic message should not appear.

If you are starting PMCD this way and the symptomatic message appears, a problem has occurred with the connection to one of the deceased PMCD's clients.

This could happen when the network connection to a remote client is lost and PMCD is subsequently terminated. The system may attempt to keep the socket open for a time to allow the remote client a chance to reestablish the connection and read any outstanding data.

The only solution in these circumstances is to wait until the socket times out and the kernel deletes it. This `netstat` command displays the status of the socket and any connections:

```
netstat -ant | grep 44321
```

If the socket is in the `FIN_WAIT` or `TIME_WAIT` state, then you must wait for it to be deleted. Once the command above produces no output, PMCD may be restarted. Less commonly, you may have another program running on your system that uses the same Internet port number (44321) that PMCD uses.

Refer to the `PCPIntro(1)` man page for a description of how to override the default PMCD port assignment using the `PMCD_PORT` environment variable.

1.2.3 Programming Performance Co-Pilot

Performance Co-Pilot (PCP) provides a systems-level suite of tools that cooperate to deliver distributed, integrated performance management services. PCP is designed for the in-depth analysis and sophisticated control that are needed to understand and manage the hardest performance problems in the most complex systems.

PCP provides unparalleled power to quickly isolate and understand performance behavior, resource utilization, activity levels and performance bottlenecks.

Performance data may be collected and exported from multiple sources, most notably the hardware platform, the operating system kernel, layered services, and end-user applications.

There are several ways to extend PCP by programming certain of its components:

- By writing a Performance Metrics Domain Agent (PMDA) to collect performance metrics from an uncharted performance domain (Chapter 2, Writing a PMDA)
- By creating new analysis or visualization tools using documented functions from the Performance Metrics Application Programming Interface (PMAPI) (Chapter 3, PMAPI–The Performance Metrics API)
- By adding performance instrumentation to an application using facilities from PCP libraries, which offer both sampling and event tracing models.

Finally, the topic of customizing an installation is covered in the chapter on customizing and extending PCP service in the Performance Co-Pilot User's and Administrator's Guide.

Contents

PCP Architecture

This section gives a brief overview of PCP architecture. For an explanation of terms and acronyms, refer to Appendix A, Acronyms.

PCP consists of numerous monitoring and collecting tools. Monitoring tools such as pmval and pminfo report on metrics, but have minimal interaction with target systems. Collection tools, called PMDAs, extract performance values from target systems, but do not provide user interfaces.

Systems supporting PCP services are broadly classified into two categories:

> **Collector**: Hosts that have the PMCD and one or more PMDAs running to collect and export performance metrics
>
> **Monitor**: Hosts that import performance metrics from one or more collector hosts to be consumed by tools to monitor, manage, or record the performance of the collector hosts

Each PCP enabled host can operate as a collector, or a monitor, or both.

Figure 1.1, "PCP Global Process Architecture" shows the architecture of PCP. The monitoring tools consume and process performance data using a public interface, the Performance Metrics Application Programming Interface (PMAPI).

Below the PMAPI level is the PMCD process, which acts in a coordinating role, accepting requests from clients, routing requests to one or more PMDAs, aggregating responses from the PMDAs, and responding to the requesting client.

Each performance metric domain (such as the operating system kernel or a database management system) has a well-defined name space for referring to the specific performance metrics it knows how to collect.

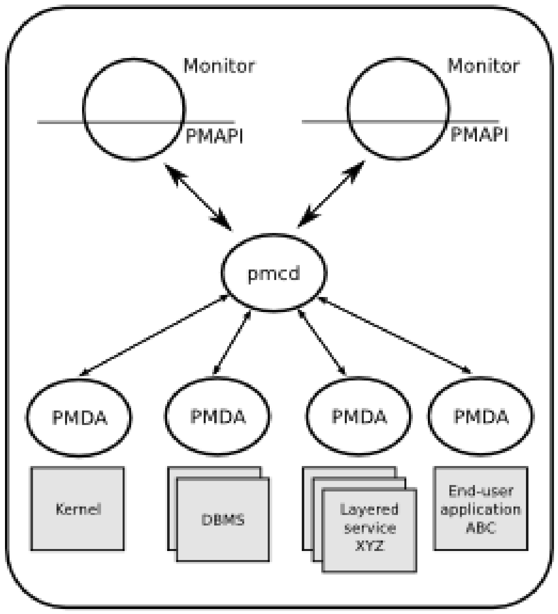

Figure 1.1. PCP Global Process Architecture

Distributed Collection

be running on the operating system for which it is collecting performance measurements; there are some notable PMDAs such as Cisco and Cluster that are exceptions, and collect performance data from remote systems.

As shown in Figure 1.2, “Process Structure for Distributed Operation”, monitoring tools communicate only with PMCD. The PMDAs are controlled by PMCD and respond to requests from the monitoring tools that are forwarded by PMCD to the relevant PMDAs on the collector host.

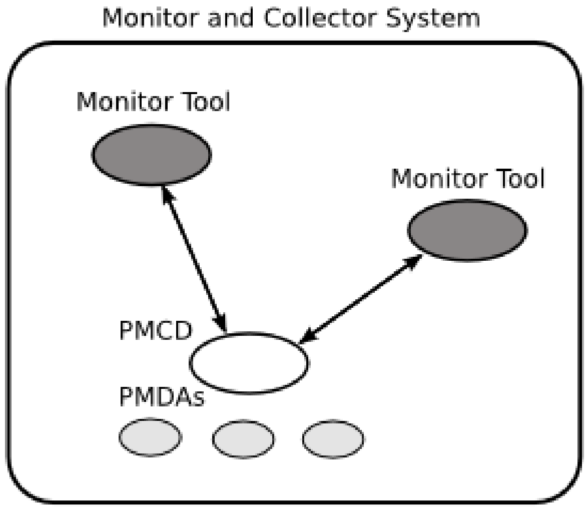

Figure 1.2. Process Structure for Distributed Operation

The host running the monitoring tools does not require any collection tools, including PMCD, since all requests for metrics are sent to the PMCD process on the collector host.

The connections between monitoring tools and PMCD processes are managed in libpcp, below the PMAPI level; see the PMAPI(3) man page. Connections between PMDAs and PMCD are managed by the PMDA functions; see the

PMDA(3) and pmcd(1) man pages. There can be multiple monitor clients and multiple PMDAs on the one host, but there may be only one PMCD process.

Name Space

Each PMDA provides a domain of metrics, whether they be for the operating system, a database manager, a layered service, or an application module. These metrics are referred to by name inside the user interface, and with a numeric Performance Metric Identifier (PMID) within the underlying PMAPI.

The PMID consists of three fields: the domain, the cluster, and the item number of the metric. The domain is a unique number assigned to each PMDA. For example, two metrics with the same domain number must be from the same PMDA. The cluster and item numbers allow metrics to be easily organized into groups within the PMDA, and provide a hierarchical taxonomy to guarantee uniqueness within each PMDA.

The Performance Metrics Name Space (PMNS) describes the exported performance metrics, in particular the mapping from PMID to external name, and vice-versa.

Distributed PMNS

Performance metric namespace (PMNS) operations are directed by default to the host or set of archives that is the source of the desired performance metrics.

In Figure 1.2, "Process Structure for Distributed Operation", both Performance Metrics Collection Daemon (PMCD) processes would respond to PMNS queries from monitoring tools by referring to their local PMNS. If different PMDAs were installed on the two hosts, then the PMNS used by each PMCD would be different, to reflect variations in available metrics on the two hosts.

Although extremely rarely used, the -n pmnsfile command line option may be used with many PCP monitoring tools to force use of a local PMNS file in preference to the PMNS at the source of the metrics.

Retrospective Sources of Performance Metrics

The distributed collection architecture described in the previous section is used when PMAPI clients are requesting performance metrics from a real-time or live source.

The PMAPI also supports delivery of performance metrics from a historical source in the form of a PCP archive log. Archive logs are created using the pmlogger utility, and are replayed in an architecture as shown in Figure 1.3, "Architecture for Retrospective Analysis".

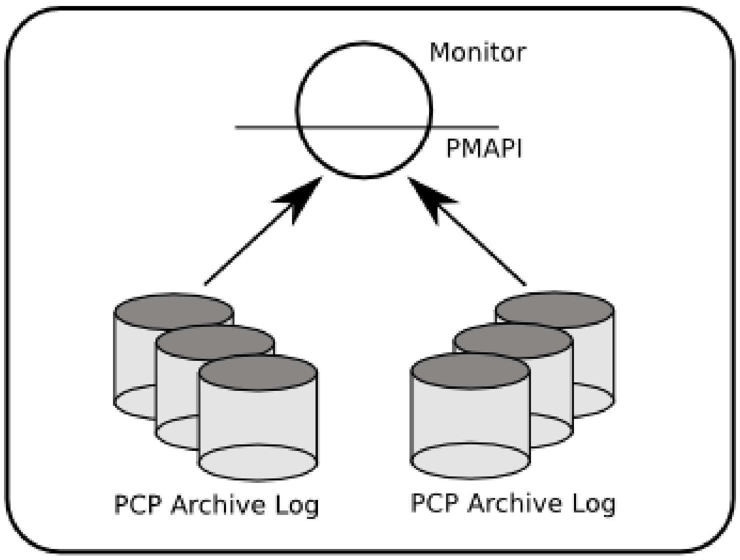

Figure 1.3. Architecture for Retrospective Analysis

Overview of Component Software

Performance Co-Pilot (PCP) is composed of text-based tools, optional graphical tools, and related commands. Each tool or command is fully documented by a man page. These man pages are named after the tools or commands they describe, and are accessible through the man command. For example, to see the pminfo(1) man page for the pminfo command, enter this command:

```
man pminfo
```

A list of PCP developer tools and commands, grouped by functionality, is provided in the following section.

Application and Agent Development

The following PCP tools aid the development of new programs to consume performance data, and new agents to export performance data within the PCP framework:

chkhelp

Checks the consistency of performance metrics help database files.

dbpmda

Allows PMDA behavior to be exercised and tested. It is an interactive debugger for PMDAs.

mmv

Is used to instrument applications using Memory Mapped Values (MMV). These are values that are communicated with pmcd instantly, and very efficiently, using a shared memory mapping. It is a program instrumentation library.

newhelp

Generates the database files for one or more source files of PCP help text.

pmapi

Defines a procedural interface for developing PCP client applications. It is the Performance Metrics Application Programming Interface (PMAPI).

pmclient

Is a simple client that uses the PMAPI to report some high-level system performance metrics. The source code for pmclient is included in the distribution.

pmda

Is a library used by many shipped PMDAs to communicate with a pmcd process. It can expedite the development of new and custom PMDAs.

pmgenmap

Generates C declarations and cpp macros to aid the development of customized programs that use the facilities of PCP. It is a program development tool.

PMDA Development

A collection of Performance Metrics Domain Agents (PMDAs) are provided with PCP to extract performance metrics. Each PMDA encapsulates domain-specific knowledge and methods about performance metrics that implement the uniform access protocols and functional semantics of the PCP. There is one PMDA for the operating system, another for process specific statistics, one each for common DBMS products, and so on. Thus, the range of performance metrics can be easily extended by implementing and integrating new PMDAs. Chapter 2, Writing a PMDA, is a step-by-step guide to writing your own PMDA.

Overview

Once you are familiar with the PCP and PMDA frameworks, you can quickly implement a new PMDA with only a few data structures and functions. This book contains detailed discussions of PMDA architecture and the integration of PMDAs into the PCP framework. This includes integration with PMCD. However, details of extracting performance metrics from the underlying instrumentation vary from one domain to another and are not covered in this book.

A PMDA is responsible for a set of performance metrics, in the sense that it must respond to requests from PMCD for information about performance metrics, instance domains, and instantiated values. The PMCD process generates requests on behalf of monitoring tools that make requests using PMAPI functions.

You can incorporate new performance metrics into the PCP framework by creating a PMDA, then reconfiguring PMCD to communicate with the new PMDA.

Building a PMDA

A PMDA interacts with PMCD across one of several well-defined interfaces and protocol mechanisms. These implementation options are described in the Performance Co-Pilot User's and Administrator's Guide.

Note: It is strongly recommended that code for a new PMDA be based on the source of one of the existing PMDAs below the `${PCP_PMDAS_DIR}` directory.

In-Process (DSO) Method

This method of building a PMDA uses a Dynamic Shared Object (DSO) that is attached by PMCD, using the platform-specific shared library manipulation interfaces such as dlopen(3), at initialization time. This is the highest performance option (there is no context switching and no interprocess communication (IPC) between the PMCD and the PMDA), but is operationally intractable in some situations. For example, difficulties arise where special access permissions are required to read the instrumentation behind the performance metrics (pmcd does not run as root), or where the performance metrics are provided by an existing process with a different protocol interface. The DSO PMDA effectively executes as part of PMCD; so great care is required when crafting a PMDA in this manner. Calls to exit(1) in the PMDA, or a library it uses, would cause PMCD to exit and end monitoring of that host. Other implications are discussed in Section 2.2.3, “Daemon PMDA”.

Daemon Process Method

Functionally, this method may be thought of as a DSO implementation with a standard main routine conversion wrapper so that communication with PMCD uses message passing rather than direct procedure calls. For some very basic examples, see the `${PCP_PMDAS_DIR}/trivial/trivial.c` and `${PCP_PMDAS_DIR}/simple/simple.c` source files.

The daemon PMDA is actually the most common, because it allows multiple threads of control, greater (different user) privileges when executing, and provides more resilient error encapsulation than the DSO method.

Note: Of particular interest for daemon PMDA writers, the `${PCP_PMDAS_DIR}/simple` PMDA has implementations in C, Perl and Python.

Client Development and PMAPI

Application developers are encouraged to create new PCP client applications to monitor, display, and analyze performance data in a manner suited to their particular site, application suite, or information processing environment.

PCP client applications are programmed using the Performance Metrics Application Programming Interface (PMAPI), documented in Chapter 3, PMAPI–The Performance Metrics API. The PMAPI, which provides performance tool developers with access to all of the historical and live distributed services of PCP, is the interface used by the standard PCP utilities.

Library Reentrancy and Threaded Applications

While the core PCP library (libpcp) is thread safe, the layered PMDA library (libpcp_pmda) is not. This is a deliberate design decision to trade-off commonly required performance and efficiency against the less common requirement for multiple threads of control to call the PCP libraries.

The simplest and safest programming model is to designate at most one thread to make calls into the PCP PMDA library.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search