Performance Co-Pilot™ User's and Administrator's Guide

007–2614–004

CONTRIBUTORS

Engineering and written contributions by Mark Goodwin, Ken McDonell, Heidi Muehlebach, Ivan Rayner, Nathan Scott, Timothy Shimmin, and Bill Tuthill.

New Features

Performance Co-Pilot™ User's and Administrator's Guide 007–2614–004

This guide contains the following new or changed information for the Performance Co-Pilot release 2.1:

- The `cron` scripts for performing PCP housekeeping task have been renamed:

New Name	Old Name
`pmlogger_check`	`cron.pmcheck`
`pmlogger_daily`	`cron.pmdaily`
`pmlogger_merge`	`cron.logmerge`
`pmsnap`	`cron.pmsnap`

- A set of parameterized `pmie` rules have been developed which are applicable to most systems. A new utility, `pmieconf`, allows these rules to be enabled or disabled, or the parameters and thresholds adjusted for a specific system (see Section 6.7, page 143).
- Extra support for running `pmie` as a daemon has been added (see Section 6.9, page 150).
- A new tool to convert arbitrary time-stamped data streams to PCP archive logs, called `pmimport`, has been added. `pmimport` currently supports the conversion of `sar/sadc` data files from the IRIX operating systems 6.2, 6.3, 6.4 and 6.5, and UNICOS 9.0.
- The new `sendmail` PMDA exports mail traffic statistics as collected by `sendmail`.
- To enable additional quality of service and availability monitoring with the `shping` PMDA, new utilities have been added to check the status of HIPPI interfaces on a system (`hipprobe`) and to interrogate the Auto-FS daemon (`autofsd_probe`).
- A new visualization utility, `xlv_vis`, has been added which displays a three-dimension bar chart of XLV volume activity.
- As an aid to creating `pmlogger` configuration files, `pmlogconf` is a new tool that allows selection of groups of commonly desired metrics and customization of `pmloggger` configurations from a simple interactive dialog.
- New capabilities have been added to assist in the estimation of PCP archive sizes. This is achieved using the `-r` option for `pmlogger` and the `-s` option for `pmdumplog`.
- Some metrics have been added to the `hotproc` PMDA. The metric `hotproc.control.config` exports the parsed one line representation of the configuration predicate and the metrics, `hotproc.predicate.*`, export the values of the variables used in the predicate.

- `pmchart` has been updated as follows:
  - Dynamic views are now supported where the view specification is computed by an application (most often `/bin/sh`) at the time the view is instantiated.
  - The menus have been slightly rearranged.
  - A new `-C` option allows `pmchart` to parse a configuration checking for errors and to exit.
  - `pmie` has been enhanced as follows:
    - Actions may now have an arbitrary number of quoted arguments.
    - Metrics with dynamic instance domains are now supported.
    - The language has been extended to allow two new operators: `match_inst` and `nomatch_inst`
    - Macro expansion can now occur anywhere in the pmie rule specifications.
  - `mlogsummary` has been extended with a `-B` option to display the distribution of values across a PCP archive in a given number of bins, with a `-p` option to adjust floating point precision and the `-I` option now reports when the minimum and maximum occurred.
  - The pcp startup script, `/etc/init.d/pcp`, now starts `pmlogger` up in the background.
- Figures and examples have been updated to reflect the new PCP names and paths.

In addition, miscellaneous editing changes were made throughout the document.

Record of Revision

Version	*Description*
004	July 1999 Revised to support the Performance Co-Pilot release 2.1 for Silicon Graphics systems running the IRIX 6.2, 6.3, 6.4, and 6.5 operating systems.

Contents

Page

Page

Page

Page

Page

Page

Page

Page

Figures

Page

Page

Tables

About This Guide

This guide describes the Performance Co-Pilot (PCP) software package of advanced performance tools for the SGI family of graphical workstations and servers.

The *Performance Co-Pilot User's and Administrator's Guide* documents both the PCP features that are embedded in the IRIX operating system and those that are in the Performance Co-Pilot (PCP) software package, which users purchase separately.

The *Performance Co-Pilot IRIX Base Software Administrator's Guide* documents the PCP features that are embedded in the IRIX operating system. This manual is a subset of the *Performance Co-Pilot User's and Administrator's Guide*.

Performance Co-Pilot provides a systems-level suite of tools that cooperate to deliver integrated performance monitoring and performance management services spanning the hardware platforms, operating systems, service layers, database management systems, and user applications.

"About This Guide" includes short descriptions of the chapters in this book, directs you to additional sources of information, and explains typographical conventions.

What This Guide Contains

This guide contains the following chapters:

- Chapter 1, page 1, provides an introduction, a brief overview of the software components, and conceptual foundations of the PCP product.
- Chapter 2, page 25, describes the basic installation and configuration steps necessary to get PCP running on your systems.
- Chapter 3, page 43, summarizes user interface components that are common to most of the graphical tools and text-based utilities that constitute the PCP monitor software.
- Chapter 4, page 61, describes the basic interactive performance monitoring tools available in PCP, including `pmchart`, `pmgadgets`, `pmkstat`, `pmdumptext`, `pmval`, `pmem`, `pminfo`, and `pmstore`.

- Chapter 5, page 95, discusses the various 3D visualization tools that are provided to enable high-level monitoring, management, and diagnosis for performance problems.
- Chapter 6, page 117, introduces the automated reasoning facilities of PCP that provide both real-time and retrospective filtering of performance data to identify adverse performance scenarios and raise alarms.
- Chapter 7, page 155, covers the PCP services and utilities that support archive logging for capturing accurate historical performance records.
- Chapter 8, page 177, presents the various options for deploying PCP functionality across systems spanning the enterprise.
- Chapter 9, page 189, describes the procedures necessary to ensure that the PCP configuration is customized in ways that maximize the coverage and quality of performance monitoring and management services.
- Appendix A, page 203, provides a comprehensive list of the acronyms used in this guide, in the man pages, and in the release notes for Performance Co-Pilot.

Audience for This Guide

This guide is written for the system administrator or performance analyst who is directly using and administering PCP applications. It is assumed that you have installed IRIS InSight for viewing online books, or have access to the *IRIX Admin* manual set, including *IRIX Admin: System Configuration and Operation*, and the *Personal System Administration Guide* as hard copy documents.

Additional Resources

The *Performance Co-Pilot Programmer's Guide* is a companion document intended for application developers who wish to use the PCP framework and services for exporting additional collections of performance metrics, or for delivering new or customized applications to enhance performance management.

Additional resources include man pages, release notes, and SGI web sites.

Man Pages

The IRIX man pages provide concise reference information on the use of IRIX commands, subroutines, and system resources. There is usually a man page for

each PCP command or subroutine. To see a list of all the PCP man pages, enter the following command:

```
man -k performance
```

To see a particular man page, supply its name to the `man` command, for example:

```
man pcp
```

The man pages are divided into the following seven sections:

(1) General commands

(2) System calls and error numbers

(3) Library subroutines

(4) File formats

(5) Miscellaneous

(6) Demos and games

(7) Special files

When referring to man pages, this guide follows a standard UNIX convention: the section number in parentheses follows the item. For example, `PMDA`(3) refers to the man page in section 3 for the `pmda` command.

Release Notes

Release notes provide specific information about the current release, available online through the `relnotes`(1) command. Exceptions to the printed and online documentation are found in the release notes. The `grelnotes` command provides a graphical interface to the release notes of all products installed on your system.

SGI Web Sites

The following Web sites are accessible to everyone with general Internet access:

`http://www.sgi.com`

The SGI general Web site, with search capability.

`http://www.sgi.com/software`

Links to Performance Co-Pilot product information.

`http://techpubs.sgi.com`

The SGI Technical Publications Library.

Obtaining Publications

To order a document, call +1 651 683 5907. SGI employees may send e-mail to `orderdsk@sgi.com`.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions Used in This Guide

These type conventions and symbols are used in this guide:

Italics	Italic typeface denotes variable entries and words or concepts being defined.
`Fixed-width type`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, keys, messages, error messages, prompts, onscreen text, and programming language structures.
`Bold fixed-width type`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
ALL CAPS	All capital letters denote environment variables, operator names, directives, defined constants, and macros in C programs.
()	Parentheses that follow function names surround function arguments or are empty if the function has no arguments; parentheses that follow IRIX commands surround man page section numbers.
[]	Brackets surround optional syntax statement arguments.
#	The pound character is the IRIX shell prompt for the superuser (`root`).

`%`	The percent character is the IRIX shell prompt for users other than the superuser.
`>>`	Two greater than characters denote the Command Monitor prompt.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. Be sure to include the title and part number of the document with your comments.

You can contact us in any of the following ways:

- Send e-mail to the following address:

  `techpubs@sgi.com`

- Send a fax to the attention of "Technical Publications" at: +1 650 932 0801.
- Use the Feedback option on the Technical Publications Library World Wide Web page:

  `http://techpubs.sgi.com`

- Call the Technical Publications Group, through the Technical Assistance Center, at: 1 800 800 4SGI.
- Send mail to the following address:

  Technical Publications
  SGI
  1600 Amphitheatre Pkwy.
  Mountain View, California 94043–1351

We value your comments and will respond to them promptly.

Introduction to Performance Co-Pilot [1]

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

The following sections are included:

- Section 1.1, page 1, covers the intended purposes of PCP.
- Section 1.2, page 5, describes PCP tools and agents.
- Section 1.3, page 13, discusses the design theories behind PCP.

1.1 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.

1.1.1 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.

1.1.2 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.

1.1.3 Unification of Performance Metric Domains

At the lowest level, performance metrics are collected and managed in autonomous performance domains such as the IRIX operating system, a database management system, 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.

1.1.4 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.

1.1.5 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.

1.1.6 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.

1.1.7 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 VCR controls, supports play, step, rewind, fast forward at variable speed processing of archived performance data.

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 VCR-like viewing controls, provides new and powerful ways to build performance tools and to review both current and historical performance data.

1.1.8 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

1.1.9 PCP Extensibility

PCP permits the integration of new performance metrics into the Performance Metrics Name Space (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 customers, 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.

1.1.10 Additional PCP Features

The following PCP features are available:

Metric coverage

The core PCP modules support export of performance metrics that include all IRIX 6.2 (and later) kernel instrumentation,

hardware instrumentation, process-level resource utilization, and activity in the PCP collection infrastructure.

The supplied agents support over 1000 distinct performance metrics, many of which can have multiple values, for example, per disk, per CPU, or per process.

Additional metrics in the layered PCP product

The PCP product extends the core modules with performance metrics that cover customizable summaries of performance metrics, `sendmail` activity and queue lengths, response time for arbitrary command execution as a quality of service measure, a dynamic subset of processes that are interesting according to user-defined criteria, environmental monitors for Challenge systems, Cisco router statistics, and application instrumentation services.

Add-on products

Additional PCP products extend the scope of performance metrics and tools to cover the following layered services:

- World Wide Web (WWW) serving
- Oracle DBMS deployments
- HPC and array environments
- SGI IRIS FailSafe platforms

The add-on products share the basic PCP operational model, APIs, architectural deployment, and protocols. Additional documentation is provided with each add-on product to describe specific installation, operation, and functional details.

1.2 Overview of Component Software

Performance Co-Pilot (PCP) is composed of text-based tools, 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
```

Many PCP tools and commands are accessible from an `Icon Catalog` on the IRIX desktop, grouped under `PerfTools`. In the `Toolchest Find` menu, choose `PerfTools`; an `Icon Catalog` appears, containing clickable PCP programs. To bring up a Web-based introduction to Performance Co-Pilot, click the `AboutPCP` icon.

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

1.2.1 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:

`dkvis`	Displays a three-dimension bar chart showing activity in the disk subsystem. It is a front-end wrapper for `pmview`.
`mpvis`	Displays a three-dimension bar chart of multiprocessor CPU utilization. It is a front-end wrapper for `pmview`.
`nfsvis`	Displays a three-dimension bar chart showing NFS (Network File System) client and server request activity, for systems on which the optional NFS software product has been installed. It is a front-end wrapper for `pmview`.
`nodevis`	Visualizes SGI Origin node statistics on platforms that support this hardware.
`osvis`	Displays three-dimension bar charts covering many aspects of system performance, including disk use, job load, memory, CPU activity, and network I/O. It is a front-end wrapper for `pmview`.
`oview`	Visualizes the performance of SGI Origin systems, showing a dynamic display of Origin node topology and performance.
`pmchart`	Displays trends over time for arbitrarily selected performance metrics from one or more hosts, or from one or more performance metric domains.

`pmdumpmineset`	Is a wrapper for `pmdumptext` that produces data files suitable for importing into the MineSet data mining product.
`pmdumptext`	Outputs the values of performance metrics collected live or from a PCP archive, as ASCII text.
`pmem`	Reports per-process memory usage statistics. Both virtual size and prorated physical memory usage are reported.
`pmgadgets`	Creates a small window containing a collection of graphical gadgets of assorted type and style, driven by performance metrics supplied by the PCP framework. Any numeric metric can be used to animate a gadget. This command is not normally invoked directly by users.
`pmgcisco`	Monitors interface throughput for Cisco routers using the `pmgadgets` tool.
`pmgevctr`	Uses `pmgadgets` to display an animated gadget that reports activity in the CPU and memory subsystems along with selected the R10K/R12K event counters and SGI Origin router metrics.
`pmgshping`	Monitors service quality and availability as measured by the `shping` PMDA using the `pmgadgets` tool.
`pmgsys`	Determines the hardware configuration of a remote or local system, constructs a suitable specification for a system-level visual monitor, and launches the `pmgadgets` tool to animate the monitor using IRIX performance metrics.
`pmie`	Evaluates predicate-action rules over performance metrics domain, for performance alarms, automated system management tasks, dynamic tuning configuration, and so on. It is an inference engine.
`pmieconf`	Creates parameterized rules to be used with Performance Co-Pilot inference engine (`pmie`).

`pminfo`	Displays information about arbitrary performance metrics available from PCP, including help text with `-T`.
`pmkstat`	Provides a text-based display of metrics that summarize system performance at a high level, suitable for ASCII logs or inquiry over a modem.
`pmlogsummary`	Calculates and reports various statistical summaries of the performance metric values from a PCP archive.
`pmprobe`	Probes for performance metric availability, values, and instances.
`pmsocks`	Allows the execution of PCP tools through a network firewall system provided `sockd` services are supported.
`pmtime`	Provides a graphical user interface for PCP applications requiring time control. This command is not normally invoked directly by users.
`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 modem.
`pmview`	Supports dynamic displays of clusters of related performance metrics as groups of utilization blocks (or towers) on a common base plane. The `pmview` tool is a generalized three-dimension (3-D) Open Inventor application. This command is not normally invoked directly by users.
`psmon`	Selects a subset of the actively running processes and launches either `pmchart` or `pmlogger` to collect per-process metrics for those processes.
`routervis`	Visualizes SGI Origin router utilization on platforms that support this hardware.
`xbowvis`	Visualizes the Crossbow (XBow) packet and error rates on platforms that support this hardware. It is a front-end wrapper for `pmview`.

`xlv_vis`	Visualizes XLV volume activity and performance.

1.2.2 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`.
`mkpmemarch`	Creates a PCP archive suitable for use with the `pmem` tool.
`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.
`pmdacisco`	Extracts performance metrics from one or more Cisco routers. It is a Performance Metrics Domain Agent (PMDA).
`pmdahotproc`	Exports performance metrics from an instance domain of processes restricted to an interesting or "hot" set. It is a PMDA.
`pmdamailq`	Exports performance metrics describing the current state of items in the `sendmail` queue. It is a PMDA.
`pmdasendmail`	Exports mail activity statistics from `sendmail`. It is a PMDA.
`pmdashping`	Exports performance metrics for the availability and quality of service (response-time) for arbitrary shell commands. It is a PMDA.
`pmdasummary`	Derives performance metrics values from values made available by other PMDAs. It is a PMDA.

`pmdatrace`	Exports transaction performance metrics from application processes that use the `pcp_trace` library. It is a PMDA.
`pmdumplog`	Displays selected state information, control data, and metric values from a PCP archive log created by `pmlogger`.
`pmimport`	Converts arbitrary time-stamped data into a PCP archive. Shipped configurations enable SAR data files from `sadc` to be translated into PCP archives.
`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 PCP archives.
`pmlogconf`	Creates or modifies `pmlogger` configuration files for most common logging scenarios. It is an interactive script.
`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.
`pmlogextract`	Reads one or more PCP archive logs and creates a temporally merged and reduced PCP archive log as output.
`pmtrace`	Provides a simple command line interface to the trace PMDA and its associated `pcp_trace` library.

1.2.3 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:

`autofsd_probe`	Probes the availability of the AutoFS mount/unmount daemon. It is used by the `shping` PMDA.
`dkmap`	Creates a map of disk real estate usage.

`dkping`	Opens the named disk for reading and checks for a response.
`dkprobe`	Initializes disk performance metrics at boot time for some IRIX versions. It may be called from `/etc/init.d/pcp`.
`hipprobe`	Probes the state of the configured HIPPI interfaces. Used by the `shping` PMDA.
`memclaim`	Allocates and holds physical memory, simulating a reduction in physical memory.
`pmbrand`	Manages the “branded” file of valid PCP licenses.
`pcp`	Summarizes that state of a PCP installation.
`pmdate`	Displays the current date and/or time, with an optional offset.
`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`).
`pmlaunch`	Contains metrics specification formats and a set of scripts for use by tools that are launching, and being launched by, other tools with no knowledge of each other. It is a configuration directory.
`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.

`pmnsmerge`	Merges multiple PMNS files together, as used by the components of the PCP.
`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.
`pmnscomp`	Compiles a PMNS in ASCII format into a more efficient binary representation.
`pmnsdel`	Removes a subtree of names from a PMNS, as used by the components of the PCP.
`pmpost`	Appends the text message to the end of the PCP notice board file (`/var/adm/pcplog/NOTICES`).
`pmrun`	Is a graphical utility for launching PCP commands with optional arguments from the IRIX desktop.
`pmsnap`	Creates performance snapshots suitable for Web publishing from PCP archives using `pmsnap`. The `pmsnap` script is intended for periodic execution via the `cron` command.
`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.

1.2.4 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. 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.

1.3 Conceptual Foundations

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

1.3.1 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 external metric names, and a mapping from external names to internal metric identifiers. See Section 1.3.6, page 17 for a description of the PMNS.

1.3.2 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 (textual) descriptions of the members of an instance domain.

Transient performance metrics (such as per-process information, per-XLV volume, and so on) 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.

1.3.3 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 an archive log 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 archive log.

Note: Performance Co-Pilot 2.*x*, and IRIX release 6.5, were developed to be completely Year 2000 compliant.

1.3.4 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.

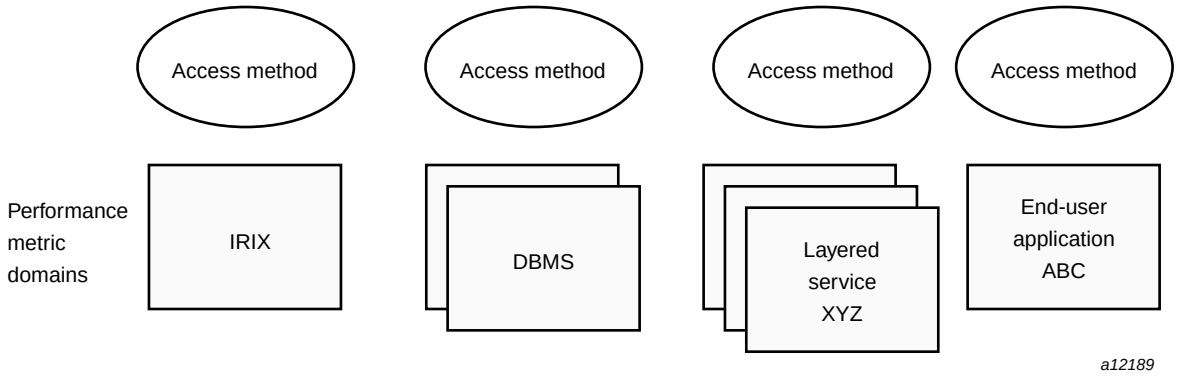

Figure 1. Performance Metric Domains as Autonomous Collections of Data

Each domain has an associated access method:

- The IRIX kernel, including `sar` data structures, per-process resource consumption, network statistics, disk activity, or memory management instrumentation.
- A DBMS such as the `V$` views and `bstat/estat` summaries for Oracle, the `tbmonitor` statistics for Informix, or the `sp_monitor` procedures for Sybase.
- A layered software service such as activity logs for a World Wide Web server or an NNTP news 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 quality agreements), or rate of computation and throughput in jobs per hour for a batch stream.
- A layered system product such as the temperature, voltage levels, and fan speeds from the environmental monitor in a Challenge system, or the length of the mail queue as reported by `mqueue`.
- 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 Metrics Domain Identifier (PMDI). 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 IRIX performance metrics domain on all hosts.

1.3.5 Distributed Collection

The performance metrics collection architecture is distributed, in the sense that any performance tool may be executing remotely. However, a PMDA must run on the system for which it is collecting performance measurements. In most cases, connecting these tools together on the collection host is the responsibility of the `pmcd` process, as shown in Figure 2.

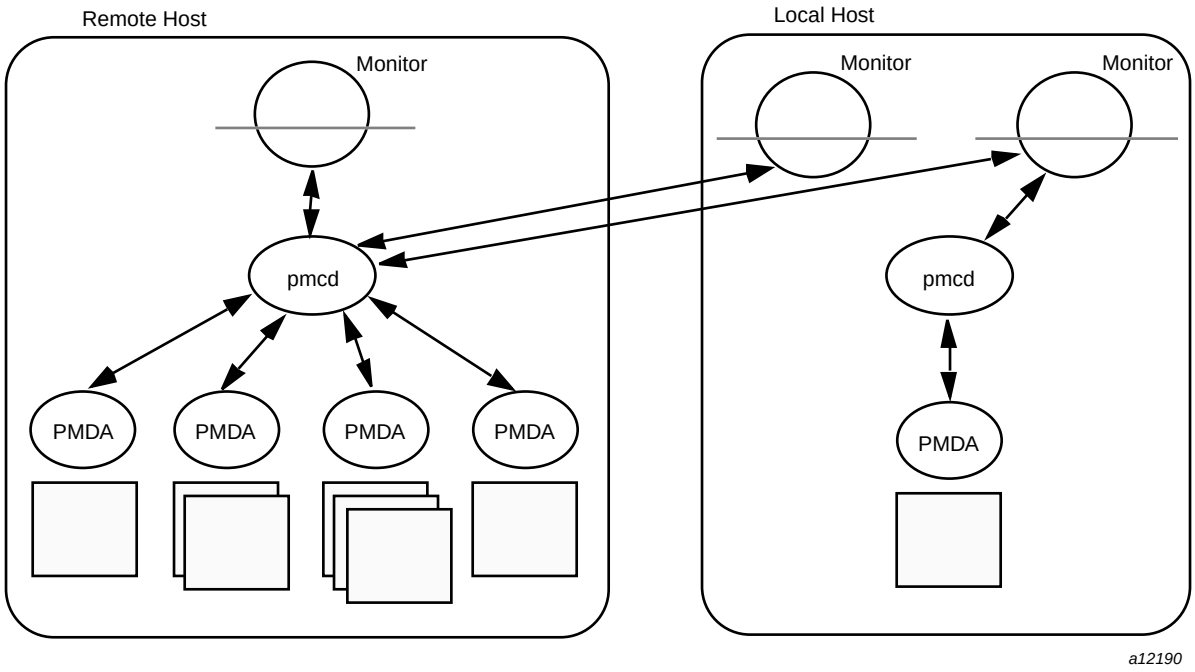

Figure 2. Process Structure for Distributed Operation

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 there may be at most one `pmcd` process.

1.3.6 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, or PMNS) maps from a hierarchy (or tree) of external names to PMIDs.

Each node in the name space 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 3 shows the nodes in a small subsection of a PMNS.

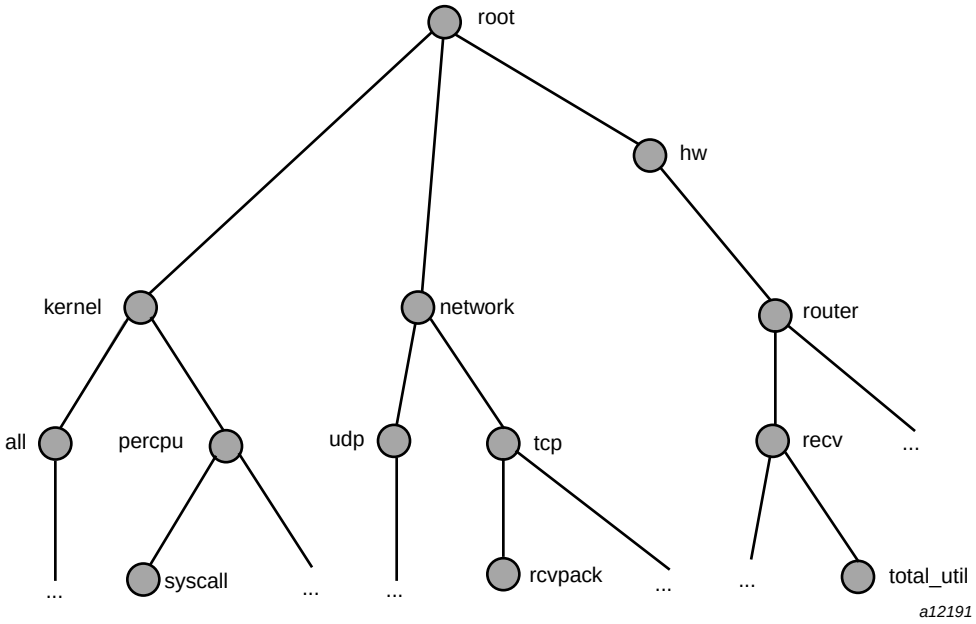

Figure 3. Small Performance Metrics Name Space (PMNS)

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

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

Although a default PMNS is shipped and updated by the components of PCP, individual users may create their own name space for metrics of interest, and all tools may use a private PMNS, rather than the default PMNS.

1.3.6.1 Distributed PMNS

In Performance Co-Pilot 1.*x* releases, the PMNS was local to the application that referred to PCP metrics by name. As of Performance Co-Pilot release 2.0, PMNS operations are directed to the host or archive that is the source of the desired performance metrics.

Distributed PMNS necessitated changes to PCP protocols between client applications and `pmcd`, and to the internal format of PCP archive files. Performance Co-Pilot release 2.*x* is compatible with earlier releases, so new PCP components operate correctly with either new or old PCP components. For example, connections to the PCP 1.*x* `pmcd`, or attempts to process a PCP archive created by a Performance Co-Pilot 1.*x* `pmlogger`, revert to using the local PMNS.

1.3.7 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 or 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 (Kbyte), or megabytes (Mbyte) 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 the performance tools applications that use the metadata when interpreting the values for performance metrics.

1.3.8 Values for Performance Metrics

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

1.3.8.1 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.

1.3.8.2 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.

Table 1. Sample Instance Identifiers for Disk Statistics

External Instance Identifier	Internal Instance Identifiers
dks1d1	131329
dks1d2	131330
dks1d3	131331
dks3d1	131841
dks3d2	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.

1.3.9 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.

1.3.10 Performance Metrics Collection System

PCP provides an infrastructure through the Performance Metrics Collection System (PMCS). It unifies the autonomous and distributed PMDAs into a cohesive pool of performance data, and provides the services required to create generalized and powerful performance tools.

The PMCS provides the framework that underpins the PMAPI, which is described in the *Performance Co-Pilot Programmer's Guide*. The PMCS is responsible for the following services on behalf of the performance tools developed on top of the PMAPI:

- Distributed namespace services
- Instance domain services
- Coordination with the processes and procedures required to control the description, collection, and extraction of performance metric values from agents that interface to the performance metric domains
- Servicing incoming requests for local performance metric values and metadata from applications running either locally or on a remote system

1.3.11 Retrospective Sources of Performance Metrics

The PMCS 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 4.

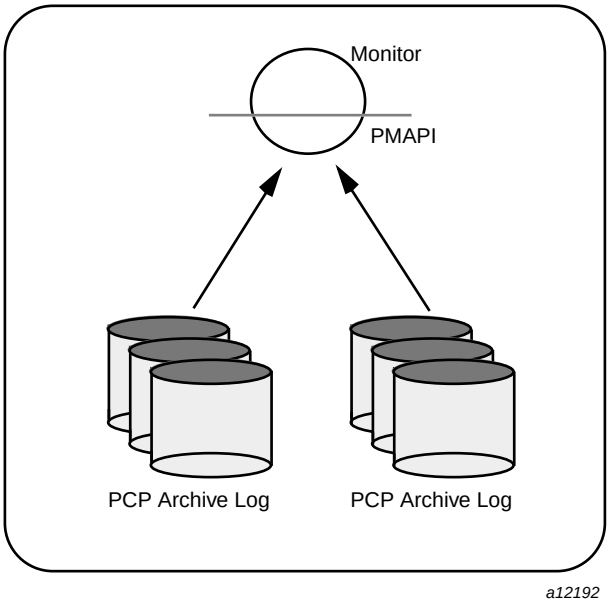

Figure 4. Architecture for Retrospective Analysis

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.

1.3.12 Product Extensibility

Much of the PCP product'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 PMCS and PMNS 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_trace`) to aid in the development of new PMDAs 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

Installing and Configuring Performance Co-Pilot [2]

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

- Section 2.1 describes the main packages of PCP software and how they must be installed on each system.
- Section 2.2, page 26, describes the additional options available for PCP.
- Section 2.3, page 27, describes the licensing issues necessary to operate PCP in a distributed computing environment.
- Section 2.4, page 27, describes the fundamentals of maintaining the performance data collector.
- Section 2.5, page 34, describes the basics of installing a new Performance Metric Domain Agent to collect metric data and pass it to the PMCD.
- Section 2.6, page 37, offers advice on problems involving the PMCD.

2.1 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.

PCP is packaged into a number of basic subsystem types that reflect the functional role of the product components. These subsystems may be installed using the `inst` or `swmgr` command:

Core — The `pcp_eoe.sw.eoe` and `pcp.sw.base` subsystems must be installed on every PCP enabled host, that is, on both PCP monitor and PCP collection systems.

Monitor — The `pcp_eoe.sw.monitor` and `pcp.sw.monitor` subsystems must be installed

	on every PCP monitor host. Subsystems `pcp_eoe.books.help` and `pcp.books.help` should be installed to provide help support for the GUI monitoring tools; see the `sgihelp`(1) man page.
Collector	No additional installation is required because the Performance Metrics Collection Daemon (`pmcd`) is in the `pcp_eoe.sw.eoe` subsystem.
Demo	The `pcp.sw.demo` subsystems provide source code for example applications and PMDAs that serve as templates for developing new modules to extend the PCP coverage of performance metrics or the capabilities of monitoring tools.
Other	The other `pcp.sw.*` subsystems provide the support for the optional PMDAs, and when required, need to be installed on the PCP collector host, and subsequently configured before they become active.
Gift	The `pcp_gifts.sw.*` subsystems provide optional applications and services that may be individually installed as required.
Documentation	The `pcp.man.*` and `pcp.books.*` subsystems provide release notes, man pages, interactive tutorials, and IRIS InSight books, and may be installed as needed.

For complete information on the installable software packages, see the Performance Co-Pilot release notes, available through the `relnotes`(1) or `grelnotes`(1) commands.

2.2 Optional Software

The capabilities of your Performance Co-Pilot (PCP) installation may be extended with added performance metrics or visual tools that are available as add-on products, sold separately from the base Performance Co-Pilot product.

For example, PCP add-on products support the following:

- World Wide Web (WWW) serving

- Oracle DBMS deployments
- HPC and array environments
- SGI IRIS FailSafe platforms

2.3 License Constraints

On Performance Co-Pilot (PCP) monitoring systems, all of the display, visualization, and automated reasoning tools are licensed using "nodelocked" FLEXlm licenses. On PCP collection systems, the Performance Metrics Collection Daemon (PMCD) is also licensed using "nodelocked" FLEXlm licenses. Refer to the PCP release notes for details.

The other PCP tools and services (for example, the Performance Metrics Domain Agents (PMDAs) or `pmlogger`) may be installed and executed without license constraints.

Some of the PCP maintenance tools for updating the Performance Metrics Name Space (PMNS), interrogating the Performance Metrics Collection System (PMCS), dumping an archive log, and so on, are not constrained by any license restrictions.

2.3.1 Using `pmbrand` to Query PCP License Capabilities

The `pmbrand` command manages the `/var/pcp/pmns/Brand` file, which contains binary information about PCP capabilities enabled by the various valid licenses on the system. If you are unsure of the license status for a particular host, `pmbrand` verifies and prints the current license information on that system, producing output similar to the following:

```
/usr/pcp/bin/pmbrand -l
Licenses for system 690794d70
     PCP Collector
     PCP Monitor
```

2.4 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.

2.4.1 Starting and Stopping the PMCD

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

```
chkconfig pmcd on
/etc/init.d/pcp 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 command

```
/etc/init.d/pcp stop
```

2.4.2 Restarting an Unresponsive PMCD

Often, 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 `/etc/init.d`. The command syntax is

```
/etc/init.d/pcp start
```

On startup, `pmcd` looks for a configuration file named `/etc/pmcd.conf`. 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:

```
killall -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.

2.4.3 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 `/var/adm/pcplog` directory, although setting the `PCPLOGDIR` environment variable before running `/etc/init.d/pcp` allows the file to be relocated.

2.4.4 PMCD Options and Configuration Files

There are two files that control PMCD operation. These are the `/etc/pmcd.conf` and `/etc/config/pmcd.options` files. The `pmcd.options` file contains the command line options used with PMCD; it is read when the daemon is invoked by `/etc/init.d/pcp`. 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.

2.4.4.1 The `pmcd.options` File

Command line options for the PMCD are stored in the `/etc/config/pmcd.options` file. The PMCD can be invoked directly from a shell prompt, or it can be invoked by `/etc/init.d/pcp` as part of the boot process. It is usual and normal to invoke it using `/etc/init.d/pcp`, 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 `/etc/init.d/pcp` is being used to start the daemon. The following options are available:

`-f`	Causes the PMCD to be run in the foreground. The PMCD is usually run in the background, as are most daemons.
`-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 `/var/adm/pcplog` or in a directory as specified by the `PCPLOGDIR` environment variable.

`-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. 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
# enable event tracing (1 for connections, 2 for PDUs, 3 for both)
# -T 3
```

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:

```
/etc/init.d/pcp start
```

2.4.4.2 The `pmcd.conf` File

When the PMCD is invoked, it reads its configuration file, which is `/etc/pmcd.conf`. This file contains entries that specify the PMDAs (Performance Metrics Domain Agents) 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 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 access

control you wish to impose. If you do not add access control rules, all access for all operations is granted to all hosts. The default `pmcd.conf` file shipped with PCP is similar to the following:

```
# Name  Id   IPC   IPC Params   File/Cmd
irix    1    dso   irix_init    libirixpmda.so
pmcd    2    dso   pmcd_init    pmda_pmcd.so
proc    3    dso   proc_init    pmda_proc.so
```

Note: Because the PMCD runs with `root` privilege, you must be very careful not to configure PMDAs in this file if you are not sure of their action. 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. If path begins with a slash (/), it is taken as an absolute path specifying the DSO; otherwise, the DSO is located in one of the directories `/usr/pcp/lib` or `/var/pcp/lib`.

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` or `AF_UNIX`. If the socket is `INET`, the word `inet` 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 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. Values for this parameter may be "text" and "binary." The text-based protocol is provided for backwards compatibility, but otherwise its use is discouraged.

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

args Optional arguments for *command*.

2.4.4.3 Controlling Access to PMCD with `pmcd.conf`

You can place this option extension in the `pmcd.conf` file to control system access to performance metric data. 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` *hostlist* `:` *operations* `;` `disallow` *hostlist* `:` *operations* `;`

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 addresses may be given in the usual four-field numeric notation. Subnet addresses may be specified using three or fewer numeric components and an asterisk as a wild card for the last component in the address.

For example, the following *hostlist* entries are all valid:

```
whizkid
gate-wheeler.eng.com
123.101.27.44
localhost
155.116.24.*
192.*
*
```

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 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 product typically reject store operations, except for selected performance metrics where the effect is benign.

For example, here is a sample access control portion of an `/etc/pmcd.conf` file:

```
allow whizkid :  all ;
allow 192.127.4.* : fetch ;
disallow gate-inet : store ;
```

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

2.5 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, `irix`, `pmcd`, and `proc`.

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 `/usr/pcp/pmdas` or `/var/pcp/pmdas`. In each directory, a `README` file describes the metrics provided by the PMDA; 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.

2.5.1 PMDA Installation

To install a PMDA you must perform a collector installation for each host on which the PMDA is required to export performance metrics. Because the PMNS

is distributed as of PCP release 2.0, it is no longer necessary to install PMDAs with their associated PMNS on PCP monitor hosts.

2.5.1.1 Installation on a PCP Collection Host

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. Move to the PMDA's directory. For example:

   `cd /var/pcp/pmdas/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 `/var/pcp/pmns/stdpmid` and the other PMDAs currently in use (listed in `/etc/pmcd.conf`). Edit `domain.h` to assign the new domain number if there is a conflict.

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 1 is illustrative of the interactions:

Example 1: PMNS Installation Output

```
You will need to choose an appropriate configuration for
installation of the ‘‘cisco’’ Performance Metrics Domain Agent (PMDA).

  collector collect performance statistics on this system
  monitor   allow this system to monitor local and/or remote systems
  both      collector and monitor configuration for this system

Please enter c(ollector) or m(onitor) or b(oth) [b] collector

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
```

2.5.2 PMDA Removal

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

PCP release 2.0, it is no longer necessary to remove PMDAs or their associated PMNS on PCP monitor hosts.

2.5.2.1 Removal on a PCP Collection Host

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

1. Log in as `root` (the superuser).
2. Move to the PMDA's directory. For example:

   ```
   cd /var/pcp/pmdas/environ
   ```

3. Enter the following command:

   ```
   ./Remove
   ```

   The following output illustrates the result:

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

2.6 Troubleshooting

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

Advice for troubleshooting the archive logging system is provided in Chapter 7, page 155.

2.6.1 Performance Metrics Name Space

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

The PMNS at the collection host is updated whenever a PMDA is installed or removed, and may also be updated when new versions of the PCP or PCP add-on products are installed. During these operations, the ASCII version of the PMNS is typically updated, then the binary version is regenerated.

2.6.2 Missing and Incomplete Values for Performance Metrics

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

2.6.2.1 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 applications 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 collection 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.

2.6.3 IRIX Metrics and the PMCD

The following issues involve the IRIX operating system and the PMCD:

- No IRIX metrics available
- Cannot connect to remote PMCD
- PMCD not reconfiguring after hang-up

- PMCD does not start

2.6.3.1 No IRIX Metrics Available

The following symptom has a known cause and resolution:

Symptom: Some of the IRIX metrics are unavailable.

Cause: PMCD (and therefore the IRIX PMDA) does not have permission to read `/dev/kmem`, or the running kernel is not the same as the kernel in `/unix`.

Resolution: Check `/var/adm/pcplog/pmcd.log`. An error message of the following form means that PMCD cannot access `/dev/kmem`.

```
kmeminit: cannot open "/dev/kmem": ...
```

Ensure that `/dev/kmem` is readable by group `sys`. For example, you should see something similar to this:

```
ls -lg /dev/kmem
crw-r-----   1 sys    1,  1 May 28 15:16 /dev/kmem
```

Restart PMCD after correcting the group and/or file permissions, and the problem should be solved.

If the running kernel is not the same as the kernel in `/unix`, the IRIX PMDA cannot access raw data in the kernel. A message like this appears in `/var/adm/pcplog/pmcd.log`:

```
kmeminit: "/unix" is not namelist for the running kernel
```

The only resolution to this is to make the running kernel the same as the one in `/unix`. If the running kernel was booted from the filesystem, then renaming files to make `/unix` the booted kernel and restarting PMCD should resolve the problem. If the running kernel was booted over the network, then PMCD cannot access the kernel's symbol table and hence the metrics

extracted by reading `/dev/kmem` directly are not available.

2.6.3.2 Cannot Connect to Remote PMCD

The following symptom has a known cause and resolution:

Symptom: A PCP client tool (such as `pmchart`, `dkvis`, 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 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.

Resolution: Establish that the PMCD on *far-away-host* is really alive, by connecting to its control port (TCP port number 4321 by default):

```
telnet far-away-host 4321
```

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:

```
/etc/init.d/pcp start
```

This response indicates the PMCD is running:

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

Interrupt the `telnet` session, increase the PMAPI timeout by setting the `PMCD_CONNECT_TIMEOUT`

environment variable to some number of seconds (60 for instance), and try the PCP tool again.

2.6.3.3 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 /etc/pmcd.conf, 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 /var/adm/pcplog/pmcd.log
```

2.6.3.4 PMCD Does Not Start

The following symptom has a known cause and resolution:

Symptom: — If the following messages appear in the PMCD log (/var/adm/pcplog/pmcd.log), consider the cause and resolution:

```
pcp[27020] Error: OpenRequestSocket(4321) 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:

```
/etc/init.d/pcp 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 -a | grep 4321
```

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 (4321) 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.

Common Conventions and Arguments [3]

This chapter deals with the user interface components that are common to most of the graphical tools and text-based utilities that make up the monitor portion of Performance Co-Pilot (PCP). These are the major sections in this chapter:

- Section 3.1, page 43, shows a picture of the `PerfTools` icons.
- Section 3.2, page 44, details some basic standards used in the development of PCP tools.
- Section 3.3, page 45, details other options to use with PCP tools.
- Section 3.4, page 47, describes the time control dialog and time-related command line options available for use with PCP tools.
- Section 3.5, page 55, describes the environment variables supported by PCP tools.
- Section 3.6, page 59, describes how to execute PCP tools that must retrieve performance data from `pmcd` on the other side of a TCP/IP security firewall.
- Section 3.7, page 60, covers some uncommon scenarios that may compromise performance metric integrity over the short term.

Many of the utilities provided with PCP conform to a common set of naming and syntactic conventions for command line arguments and options. This section outlines these conventions and their meaning. The options may be generally assumed to be honored for all utilities supporting the corresponding functionality.

In all cases, the man pages for each utility fully describe the supported command arguments and options.

Command line options are also relevant when starting PCP applications from the desktop using the `Alt` double-click method. This technique launches the `pmrun` program to collect additional arguments to pass along when starting a PCP application.

3.1 `PerfTools` Icon Catalog

The conventions and arguments described in this chapter are common to all tools and utilities in the `PerfTools Icon Catalog` group, shown in Figure 5.

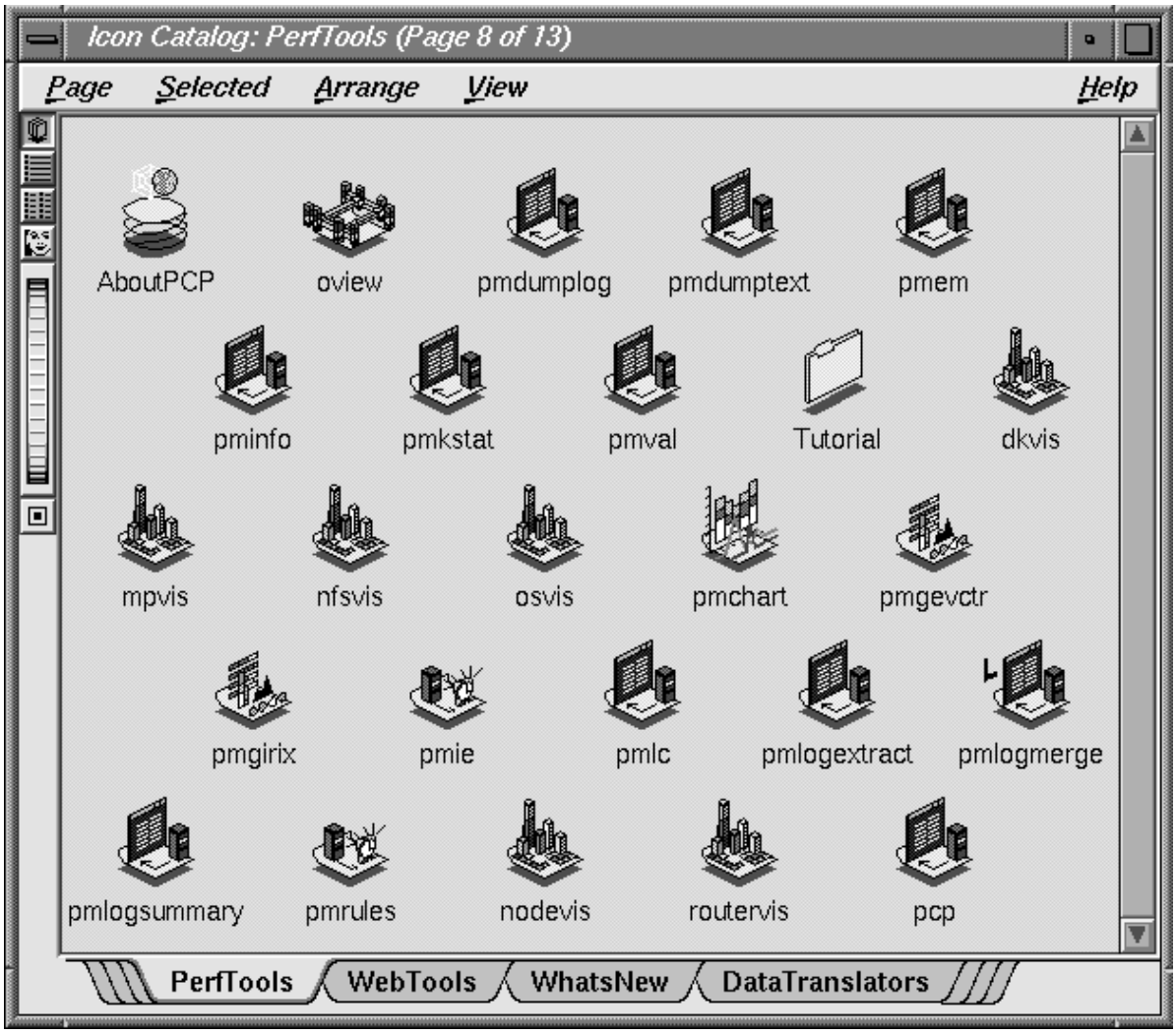

Figure 5. `PerfTools Icon Catalog` Group

3.2 Alternate Metric Source Options

The default source of performance metrics is from `pmcd` on the local host. This section describes how to obtain metrics from sources other than the default.

3.2.1 Fetching Metrics from Another Host

The option `-h` *host* directs any PCP utility (such as `pmchart` or `dkvis`) to make a connection with the `pmcd` instance running on *host*. Once established, this connection serves as the principal real-time source of performance metrics and metadata.

3.2.2 Fetching Metrics from an Archive Log

The option `-a` *archive* directs the utility to treat the PCP archive logs with base name *archive* as the principal source of performance metrics and metadata.

PCP archive logs are created with `pmlogger`. Most PCP utilities operate with equal facility for performance information coming from either a real-time feed via `pmcd` on some host, or for historical data from a PCP archive log. For more information on archive logs and their use, see Chapter 7, page 155.

The base name (`archive`) of the PCP archive log used with the `-a` option implies the existence of the files created automatically by `pmlogger`, as listed in Table 2.

Table 2. Physical Filenames for Components of a PCP Archive Log

Filename	Contents
`archive.`*index*	Temporal index for rapid access to archive contents
`archive.`*meta*	Metadata descriptions for performance metrics and instance domains appearing in the archive
`archive.N`	Volumes of performance metrics values, for `N` = 0,1,2,...

Some tools are able to concurrently process multiple PCP archive logs (for example, for retrospective analysis of performance across multiple hosts), and accept either multiple `-a` options or a comma separated list of archive names following the `-a` option.

Note: The `-h` and `-a` options are mutually exclusive in all cases.

3.3 General PCP Tool Options

The following sections provide information relevant to most of the PCP tools. It is presented here in a single place for convenience.

3.3.1 Common Directories and File Locations

The following files and directories are used by the PCP tools as repositories for option and configuration files and for binaries:

`/etc/pmcd.conf`	Configuration file for Performance Metrics Collection Daemon (PMCD).
`/usr/etc/pmcd`	The PMCD binary.
`/etc/config/`	The `pmcd.options` file contains command line options for `pmcd` .The `pmlogger.options` file contains command line options for `pmlogger` launched from `/etc/init.d/pcp`.
`/etc/init.d/pcp`	The PMCD startup script.
`/usr/sbin`	Contains PCP tools such as `pmkstat`, `pminfo`, and `oview`.
`/usr/pcp`	Shareable PCP-specific files and repository directories.
`/var/pcp`	Non-shareable (that is, per-host) PCP specific files and repository directories. There are some symbolic links from the `/usr/pcp` directory hierarchy pointing into the `/var/pcp` directory hierarchy.
`/usr/pcp/bin`	Contains PCP tools that are typically not executed directly by the end user such as `pmbrand`, `pmnscomp`, and `pmlogger`.
`/usr/pcp/lib`	Contains miscellaneous PCP libraries and executables.
`/var/pcp/pmdas`	Contains Performance Metric Domain Agents, one directory per PMDA.
`/usr/pcp/pmdas`	An alternate repository for some PMDAs. Certain entries here are symbolic links into `/var/pcp/pmdas`.
`/var/pcp/config`	Contains configuration files for PCP tools, typically with one directory per tool.
`/usr/pcp/demos`	Contains demonstration data files and example programs.
`/var/pcp/Tutorial`	Contains a PCP Tutorial, in HTML format.

`/var/adm/pcplog`	By default contains diagnostic and trace log files generated by `pmcd` and PMDAs. Also, the PCP archive logs are managed in one directory per logged host below here.
`/var/pcp/pmns`	Contains files and scripts for the Performance Metrics Name Space.

3.3.2 Alternate Performance Metric Name Spaces

The Performance Metrics Name Space (PMNS) defines a mapping from a collection of external names for performance metrics (convenient to the user) into corresponding internal identifiers (convenient for the underlying implementation).

The distributed PMNS used in PCP 2.x avoids most requirements for an alternate PMNS, because clients' PMNS operations are supported at the Performance Metrics Collection Daemon (PMCD) or by means of PMNS data in a PCP archive log. The distributed PMNS is the default, but alternates may be specified using the `-n` *namespace* argument to the PCP tools. When a PMNS is maintained on a host, it is likely to reside in the `/var/pcp/pmns` directory.

Refer to the `pmns`(4) and `pmnscomp`(1) man pages for details of PMNS structure and creation.

3.4 Time Duration and Control

The periodic nature of sampling performance metrics and refreshing the displays of the PCP tools makes specification and control of the temporal domain a common operation. In the following sections, the services and conventions for specifying time positions and intervals are described.

3.4.1 Performance Monitor Reporting Frequency and Duration

Many of the performance monitoring utilities have periodic reporting patterns. The `-t` *interval* and `-s` *samples* options are used to control the sampling (reporting) interval, usually expressed as a real number of seconds (*interval*), and the number of *samples* to be reported, respectively. In the absence of the `-s` flag, the default behavior is for the performance monitoring utilities to run until they are explicitly stopped.

The *interval* argument may also be expressed in terms of minutes, hours, or days, as described in the `PCPIntro`(1) man page.

3.4.2 Time Window Options

The following options may be used with most PCP tools (typically when the source of the performance metrics is a PCP archive log) to tailor the beginning and end points of a display, the sample origin, and the sample time alignment to your convenience.

The `-S`, `-T`, `-O` and `-A` command line options are used by PCP applications to define a time window of interest.

`-S` *duration*

The start option may be used to request that the display start at the nominated time. By default, the first sample of performance data is retrieved immediately in real-time mode, or coincides with the first sample of data in a PCP archive log in archive mode. For archive mode, the `-S` option may be used to specify a later time for the start of sampling. By default, if *duration* is an integer, the units are assumed to be seconds.

To specify an offset from the beginning of a PCP archive (in archive mode) simply specify the offset as the *duration*. For example, the following entry retrieves the first sample of data at exactly 30 minutes from the beginning of a PCP archive.

```
-S 30min
```

To specify an offset from the end of a PCP archive, prefix the *duration* with a minus sign. In this case, the first sample time precedes the end of archived data by the given *duration*. For example, the following entry retrieves the first sample exactly one hour preceding the last sample in a PCP archive.

```
-S -1hour
```

To specify the calendar date and time (local time in the reporting time zone) for the first sample, use the `ctime` syntax preceded by an "at" sign

(@). For example, the following entry specifies the date and time to be used.

```
-S '@ Mon Mar 4 13:07:47 1996'
```

Note that this format corresponds to the output format of the `date` command for easy "cut and paste." However, be sure to enclose the string in quotes so it is preserved as a single argument for the PCP tool.

For more complete information on the date and time syntax, see the `PCPIntro`(1) man page.

`-T` *duration*

The terminate option may be used to request that the display stop at the time designated by *duration*. By default, the PCP tools keep sampling performance data indefinitely (in real-time mode) or until the end of a PCP archive (in archive mode). The `-T` option may be used to specify an earlier time to terminate sampling.

The interpretation for the *duration* argument in a `-T` option is the same as for the `-S` option, except for an unsigned time interval that is interpreted as being an offset from the start of the time window as defined by the default (now for real time, else start of archive) or by a `-S` option. For example, these options define a time window that spans 45 minutes, after an initial offset (or delay) of 1 hour:

```
-S 1hour -T 45mins
```

`-O` *duration*

By default, samples are fetched from the start time (see the description of the `-S` option) to the terminate time (see the description of the `-T` option). The offset `-O` option allows the specification of a time between the start time and the terminate time where the tool should position its initial sample time. This option is useful when initial attention is focused at some point within a larger time window of interest, or when one PCP tool wishes to launch another PCP tool with a common current point of time within a shared time window.

The *duration* argument accepted by `-O` conforms to the same syntax and semantics as the *duration* argument for `-T`. For example, these options specify that the initial position should be the end of the time window:

```
-O -0
```

This is most useful with `pmchart`(1) to display the tail-end of the history up to the end of the time window.

`-A` *alignment*

By default, performance data samples do not necessarily happen at any natural unit of measured time. The `-A` switch may be used to force the initial sample to be on the specified *alignment*. For example, these three options specify alignment on seconds, half hours, and whole hours:

```
-A 1sec
-A 30min
-A 1hour
```

The `-A` option advances the time to achieve the desired alignment as soon as possible after the start of the time window, whether this is the default window, or one specified with some combination of `-A` and `-O` command line options.

Obviously the time window may be overspecified by using multiple options from the set `-t`, `-s`, `-S`, `-T`, `-A`, and `-O`. Similarly, the time window may shrink to nothing by injudicious choice of options.

In all cases, the parsing of these options applies heuristics guided by the principal of "least surprise"; the time window is always well-defined (with the end never earlier than the start), but may shrink to nothing in the extreme.

3.4.3 Time Zone Options

All utilities that report time of day use the local time zone by default. The following time zone options are available:

`-z`

Forces times to be reported in the time zone of the host that provided the metric values (the PCP

collector host). When used in conjunction with `-a` and multiple archives, the convention is to use the time zone from the first named archive.

`-Z` *timezone* — Sets the TZ variable to a time zone string, as defined in `environ`(5) , for example, `-Z UTC` for universal time.

3.4.4 PCP Live Time Control

The `pmtime PCP Live Time Control` dialog, shown in Figure 6, is invoked through the PCP tools when you select the `Show Time Control` option from the `Options` menu of most PCP tools. The dialog may also be exposed by selecting the "time control state" button at the bottom left-hand corner of the `pmchart` display or the top left-hand corner of a 3D performance scene displayed with the `pmview` or `oview` tools.

For more information on the "time control state" button, see the `pmview`(1), `pmchart`(1), `oview`(1), or `pmtime`(1) man page.

If the PCP tool is displaying performance metrics from a real-time source, the `pmtime` dialog looks similar to that shown in Figure 6.

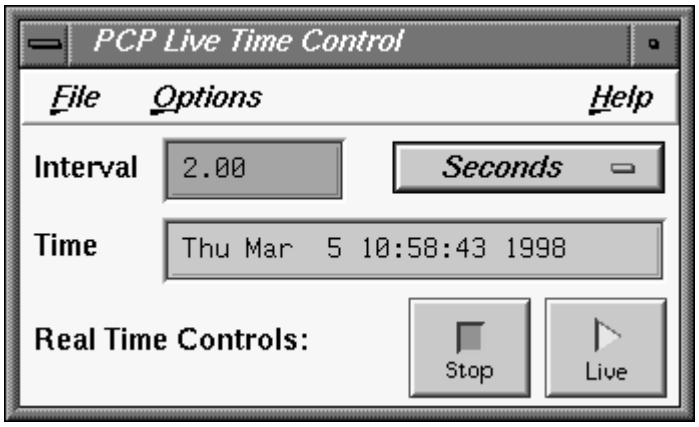

Figure 6. `pmtime PCP Live Time Control` Dialog

This dialog can be used to set the sample interval and units; the latter may be in milliseconds, seconds, minutes, hours, days, or weeks.

To change the units, select the measurement of time you want from the `Options` menu (labelled `Seconds` in Figure 6).

To change the interval, enter the new value in the Interval text box, and press Enter. All PCP tools attached to the `pmtime` control dialog are notified of the new interval, and will update their displays immediately to reflect the new sampling rate.

3.4.5 Creating a PCP Archive

The ability to start and stop recording of performance activity is available from the `pmchart`, `pmview`, and `oview` windows using the `File -> Record` option from the menu bar. See Section 4.1.5, page 75, for information about the `pmchart` interface.

Alternatively use `pmlogger` directly, as described in Chapter 7, page 155.

3.4.6 PCP Archive Time Control

The ability to provide retrospective performance analysis in the PCP framework is provided by making the monitor tools able to deal interchangeably with real-time sources of performance metrics and PCP archive logs. For more information on archive logging, see Chapter 7, page 155.

When a PCP tool is displaying performance metrics from a PCP archive log, and the `pmtime Archive Time Control` dialog is exposed, it looks similar to that shown in Figure 7.

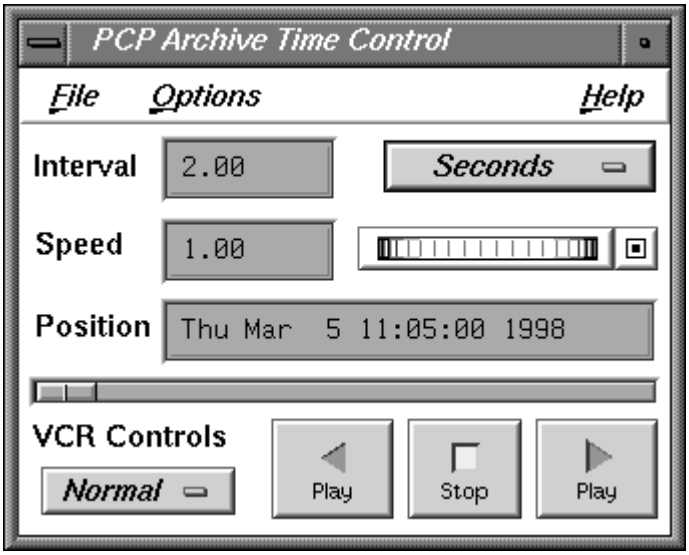

Figure 7. `pmtime` `PCP Archive Time Control` Dialog

As with the live `pmtime` dialog, the user may change the update interval; however, a number of other controls are available:

- The `VCR Controls` option menu may be used to change the mode of time advance between `Normal`, `Step`, and `Fast`.
  - In `Normal` mode, the time advances with the elapsed time per sample being equal to the current `Interval` (divided by `Speed`).
  - In `Step` mode, each selection of one of the direction buttons advances the time by the current `Interval`.
  - In `Fast` mode, the time advances by the `Interval` without any added delay.
- The `Speed` text box and associated thumb wheel may be used to make the rate of time advance in `Normal` mode either slower (`Speed` < 1) or faster (`Speed` > 1) than real time.
- The `Position` text box shows the current time within the PCP archive log. The `Position` may be changed either by advancing the time using the `VCR Controls` buttons (`Play`, `Step`, `Rewind`, `Fast Fwd`, or `Stop`), or by

modifying the Position text box (and pressing Enter), or by moving the slider below the Position text box.

- The VCR Controls motion buttons allow time to be advanced forward or backward, or stopped.

The menus of pmtime Archive Time Control provide the following additional features:

3.4.6.1 File Menu

The File menu supports the following option:

Hide	Hides the dialog; the PCP tools provide their own menu options or time control icon that may be used to reexpose the pmtime dialog.

3.4.6.2 Options Menu

The Options menu supports three options:

Timezone	Selects an alternative time zone for all displayed dates and times; all PCP tools attached to the pmtime control are notified of the new time zone. Because the UTC time zone is universal, it is useful when several archives or live sources of data are being displayed in multiple instances of the tools, and comparisons between performance metrics are required to be temporally correlated. Whenever a new source of metrics is opened, whether an archive or live, the time zone at that source of metrics is added to the list in the Options menu. The default time zone is that of the local host where the tool is being run.
Show Bounds...	Exposes the Archive Time Bounds dialog, shown in Figure 8. This dialog shows the current time window that defines the earliest and latest time for which performance may be displayed from the current archives.

Figure 8. `pmtime` `Archive Time Bounds` Dialog

`Detail` For output fields, selectively includes or excludes the year in the date or milliseconds in time. The year is shown by default, milliseconds are not.

3.5 PCP Environment Variables

The following environment variables are recognized by PCP (these definitions are also available on the `PCPIntro`(1) man page):

`PCP_COUNTER_WRAP`

Many of the performance metrics exported from PCP agents expect that counters increase monotonically. Under some circumstances, one value of a metric may be smaller than the previously fetched value. This can happen when a counter of finite precision overflows, when the PCP agent has been reset or restarted, or when the PCP agent exports values from an underlying instrumentation that is subject to asynchronous discontinuity.

If set, the `PCP_COUNTER_WRAP` environment variable indicates that all such cases of a decreasing counter should be treated as a counter overflow; and hence the values are assumed to have

wrapped once in the interval between consecutive samples. Counter wrapping was the default in versions before the PCP 1.3 release.

PCP_LICENSE_NOWARNING

Many of the PMAPI client programs require that a valid software license be present on the host on which the client is running (the license is node-locked). In the case that such a valid license is present, but is due to expire within the next 30 days, a message or popup notifier appears informing the user of this condition. These warnings can be disabled by setting this variable in the environment.

PCP_LOGDIR

Many PCP utilities create diagnostic and trace log files, and the default locations are below the `/var/adm/pcplog` directory. Setting the `PCP_LOGDIR` variable overrides the default directory. If `PCP_LOGDIR` is unset, the `PCP_LOGDIR` variable is treated as an alias and used if set, to provide backwards compatibility with earlier PCP releases.

PCP_STDERR

Specifies whether `pmprintf()` error messages are sent to standard error, an `xconfirm` dialog box, or to a named file; see the `pmprintf`(3) man page. Messages go to standard error if `PCP_STDERR` is unset or set without a value. If this variable is set to `DISPLAY`, then messages go to an `xconfirm` dialog box; see the `xconfirm`(1) man page. Otherwise, the value of `PCP_STDERR` is assumed to be the name of an output file.

PCP_TRACE_HOST

The `pmdatrace` library routines use this variable when connecting to the `trace` PMDA to determine on which host it is running; see the `pmdatrace`(3) man page.

PCP_TRACE_PORT

This variable is used by both the `trace` PMDA and client programs using the `pmdatrace` library to obtain the Internet port through which the client programs and the PMDA communicate; see the `pmdatrace`(3) man page.

PCP_TRACE_TIMEOUT

When pmdatrace client programs are connecting to the trace PMDA, this variable can be set to specify how long the clients should wait before cancelling their attempt to connect with the PMDA; see the pmdatrace(3) man page.

PMCD_CONNECT_TIMEOUT

When attempting to connect to a remote pmcd on a system that is booting or at the other end of a slow network link, some PMAPI routines could potentially block for a long time until the remote system responds. These routines abort and return an error if the connection has not been established after some specified interval has elapsed. The default interval is 5 seconds. This may be modified by setting this variable in the environment to a larger number of seconds for the desired time out. This is most useful in cases where the remote host is at the end of a slow network, requiring longer latencies to establish the connection correctly.

PMCD_PORT

This TCP/IP port is used by pmcd to create the socket for incoming connections and requests. The default is port number 4321, which you may override by setting this variable to a different port number. If a non-default port is in effect when pmcd is started, then every monitoring application connecting to that pmcd must also have this variable set in its environment before attempting a connection.

PMCD_RECONNECT_TIMEOUT

When a monitor or client application loses its connection to a pmcd, the connection may be reestablished by calling the pmReconnectContext() PMAPI function. However, attempts to reconnect are controlled by a back-off strategy to avoid flooding the network with reconnection requests. By default, the back-off delays are 5, 10, 20, 40, and 80 seconds for consecutive reconnection requests from a client (the last delay is repeated for any further attempts after the last delay in the list). Setting this environment variable to a comma-separated list of positive integers redefines the back-off delays. For example, setting the delays to **1,2** will back off for 1 second, then back off every 2 seconds thereafter.

PMCD_REQUEST_TIMEOUT

For monitor or client applications connected to pmcd, there is a possibility of the application hanging on a request for performance metrics or metadata or help text. These delays may become severe if the system running pmcd crashes or the network connection is lost or the network link is very slow. By setting this environment variable to a real number of seconds, requests to pmcd timeout after the specified number of seconds. The default behavior is to wait 10 seconds for a response from every pmcd for all applications.

PMDA_PATH

This environment variable may be used to modify the search path used by pmcd and pmNewContext() (for PM_CONTEXT_LOCAL contexts) when searching for a daemon or DSO PMDA. The syntax follows the syntax for shell PATH: a colon-separated list of directories. The default search path is /var/pcp/lib:/usr/pcp/lib.

PM_LAUNCH_PATH

A launching tool searches for its script in the directory specified by this variable, rather than /var/pcp/config/pmlaunch; see the pmlaunch(5) man page.

PMLOGGER_PORT

This environment variable may be used to change the base TCP/IP port number used by pmlogger to create the socket to which pmlc instances try to connect. The default base port number is 4330. If used, this variable should be set in the environment before pmlogger is executed. If pmlc and pmlogger are on different hosts, then obviously PMLOGGER_PORT must be set to the same value in both places.

PMNS_DEFAULT

If set, this value is interpreted as the full pathname to be used as the default PMNS for pmLoadNameSpace(). Otherwise, the

default PMNS is located at `/var/pcp/pmns/root` for base PCP installations.

3.6 Running PCP Tools through a Firewall

In some production environments, the Performance Co-Pilot (PCP) monitoring hosts are on one side of a TCP/IP firewall, and the PCP collector hosts may be on the other side.

If the firewall service is being provided by a product that supports the `sockd` (SOCKS) protocols for packet forwarding through the firewall, then the PCP tool `pmsocks` may be used; see the `pmsocks`(1) man page. Otherwise it is necessary to arrange for packet forwarding to be enabled for those TCP/IP ports used by PCP, namely 4321 (or the value of the `PMCD_PORT` environment variable) for connections to `pmcd` and a finite range of consecutive port numbers starting at 4330 (or the value of the `PMLOGGER_PORT` environment variable) to allow `pmlc` connections to `pmlogger` instances.

3.6.1 The `pmsocks` Command

The `pmsocks` command and its related files and scripts allow PCP clients running on hosts located on the internal side of a TCP/IP `sockd` firewall system to monitor remote hosts on the other side of the firewall system. The basic syntax is as follows, where *tool* is an arbitrary PCP application, typically a monitoring tool:

`pmsocks` *tool args*

The `pmsocks` script prepares the necessary environment variables and then executes the PCP tool specified in *tool* across the firewall. For example, this command runs `dkvis` with metrics fetched from *remotehost* on the other side of the firewall:

```
pmsocks dkvis -h remotehost
```

The configuration file is`/etc/pcp_socks.conf`, and the network-specific information in this file is set to correspond with your network. Complete information on this customization can be found in the `pmsocks`(1) man page.

3.7 Transient Problems with Performance Metric Values

Sometimes the values for a performance metric as reported by a PCP tool appear to be incorrect. This is typically caused by transient conditions such as metric wraparound or time skew, described below. These conditions result from design decisions that are biased in favor of lightweight protocols and minimal resource demands for PCP components.

In all cases, these events are expected to occur infrequently, and should not persist beyond a few samples.

3.7.1 Performance Metric Wraparound

Performance metrics are usually expressed as numbers with finite precision. For metrics that are cumulative counters of events or resource consumption, the value of the metric may occasionally overflow the specified range and wraparound to zero.

Because the value of these counter metrics is computed from the rate of change with respect to the previous sample, this may result in a transient condition where the rate of change is an unknown value. If the `PCP_COUNTER_WRAP` environment variable is set, this condition is treated as an overflow, and speculative rate calculations are made. In either case, the correct rate calculation for the metric returns with the next sample.

3.7.2 Time Dilation and Time Skew

If a PMDA is tardy in returning results, or the PCP monitoring tool is connected to `pmcd` via a slow or congested network, an error might be introduced in rate calculations due to a difference between the time the metric was sampled and the time `pmcd` sends the result to the monitoring tool.

In practice, these errors are usually so small as to be insignificant, and the errors are self-correcting (not cumulative) over consecutive samples.

A related problem may occur when the system time is not synchronized between multiple hosts, and the time stamps for the results returned from `pmcd` reflect the skew in the system times. In this case, it is recommended that either `timeslave` or `timed` be used to keep the system clocks on the collector systems synchronized; see the `timed`(1M) man page.

Monitoring System Performance [4]

This chapter describes the performance monitoring tools available in Performance Co-Pilot (PCP). This product provides a group of commands and tools for measuring system performance. Each tool is described completely by its own man page. The man pages are accessible through the `man` command. For example, the man page for the tool `pmchart` is viewed by entering the following command:

```
man pmchart
```

The following major sections are covered in this chapter:

- Section 4.1, page 62, describes `pmchart`, a useful charting tool that graphically monitors system performance.
- Section 4.2, page 81, presents `pmgadgets`, a graphical tool that displays system performance in a small area.
- Section 4.3, page 84, discusses `pmkstat`, a utility that provides a periodic one-line summary of system performance.
- Section 4.4, page 86, discusses `pmdumptext`, a utility that shows the current values for named performance metrics.
- Section 4.5, page 86, describes `pmval`, a utility that displays performance metrics in ASCII tables.
- Section 4.6, page 88, discusses `pmem`, a utility that reports per-process memory usage statistics.
- Section 4.7, page 89, describes `pminfo`, a utility that displays information about performance metrics.
- Section 4.8, page 93, describes the use of the `pmstore` utility to arbitrarily set or reset selected performance metric values.

Further monitoring tools covering performance visualization and automated reasoning about performance are described in Chapter 5 and Chapter 6.

The following sections describe the various graphical and text-based PCP tools used to monitor local or remote system performance.

4.1 The pmchart Tool

The pmchart utility supports interactive selection and plotting of trends over time for arbitrarily selected performance metrics from one or more hosts and one or more domains of performance metrics. First, you enter the following command:

```
pmchart
```

You then see the Performance Co-Pilot Chart window shown in Figure 9.

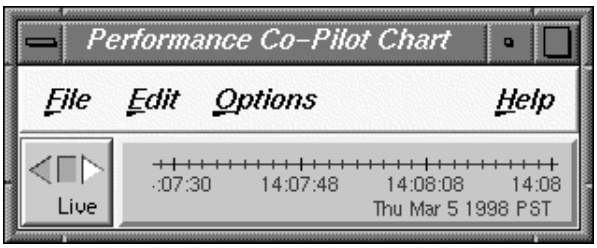

Figure 9. pmchart Performance Co-Pilot Chart Window

Normally, pmchart operates in *live* mode where performance metrics are fetched in real time and plotted against a time axis. The user can choose performance metrics and monitor the current values for these metrics from any host that is accessible on the network and has the pmcd server running.

When launched with the -a command line option, pmchart can also replay PCP archive logs of performance metrics created by pmlogger.

The man page for pmchart explains how to configure charts based on performance metrics, using either the Open View option of the File menu or the New Plot option of the File menu. Once charts have been configured and applied, the charts are placed in an expanded Performance Co-Pilot Chart window, as shown in Figure 10.

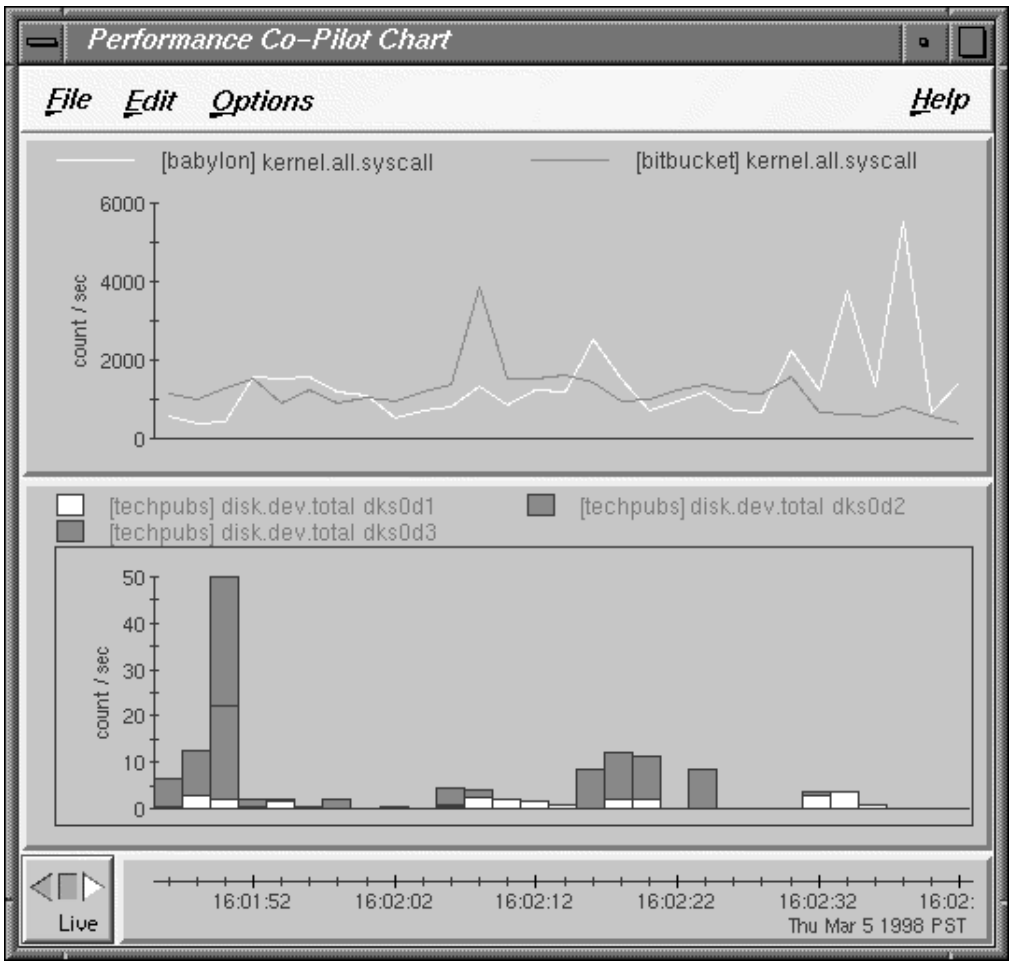

Figure 10. Two Charts and Metrics from Three Hosts in `pmchart`

All metrics in the Performance Metrics Name Space (PMNS) with numeric value semantics can be graphed. By default, `pmchart` initially allows the user to select metrics to be plotted from the local host. However, the graphical user interface allows other hosts or archives to be chosen at any time as alternate sources of performance metrics and all metrics (independent of their source) are plotted on a common time axis.

For horizontal lines at major tick marks, see Section 4.1.3, page 67.

The `-h` command line option nominates an alternate default host to be used in preference to the local host.

The `-a` command line option may be used to start `pmchart` processing performance metrics from one or more PCP archive logs. The first named archive becomes the default source of performance metrics. This mode is particularly useful for retrospective comparisons and for postmortem analysis of performance problems, where a remote system is not directly accessible or a performance analyst is not available on site.

The `pmchart` utility examines the semantics of selected metrics, and where sensible, uses the metadata provided by the Performance Metrics Collection Subsystem (PMCS) to convert fetched metric values to a rate before plotting. In the case where different metrics are plotted in the same chart (for example, against a common Y-axis), the metrics must have the same dimension (taking into account any automatic rate conversion), but `pmchart` may scale metric values where necessary, to produce comparable values with common units and scale.

When replaying archive logs, the user may interactively control the current replay time, direction of replay, and replay rate, using the PCP time control dialog, as described in the Section 3.4.

4.1.1 Mouse Controls

The `pmchart` tool uses the mouse buttons as follows:

Left	The primary mouse button may be used to select the current chart by clicking anywhere in a specific chart. The current chart always has a border drawn around the graph area and its legend of metric names rendered in red. The `Edit` menu contains a variety of choices that operate only on the current chart. This mouse button also interacts with menus and dialog boxes in the usual manner.
Middle	The middle mouse button is unused.
Right	The secondary mouse button may be used to display metric values in a dialog box. Click this mouse button in the graph drawing area of any chart to display information about the nearest metric and its value at that point as plotted. The `Metric Value Information` dialog box

remains visible until you dismiss it, and can be refreshed with new metric values by clicking this mouse button again, or updated automatically using the `Show most Recent` toggle button.

4.1.2 `pmchart Select Performance View`

A *view* in `pmchart` is a predefined collection of charts, typically constructed to display some common performance scenario. Default views are included in the PCP distribution, others are part of the various PCP add-on products, and others may be created by the `pmchart` end user. The `Open View...` option in the `File` menu launches a `Select Performance View` dialog box similar to Figure 11.

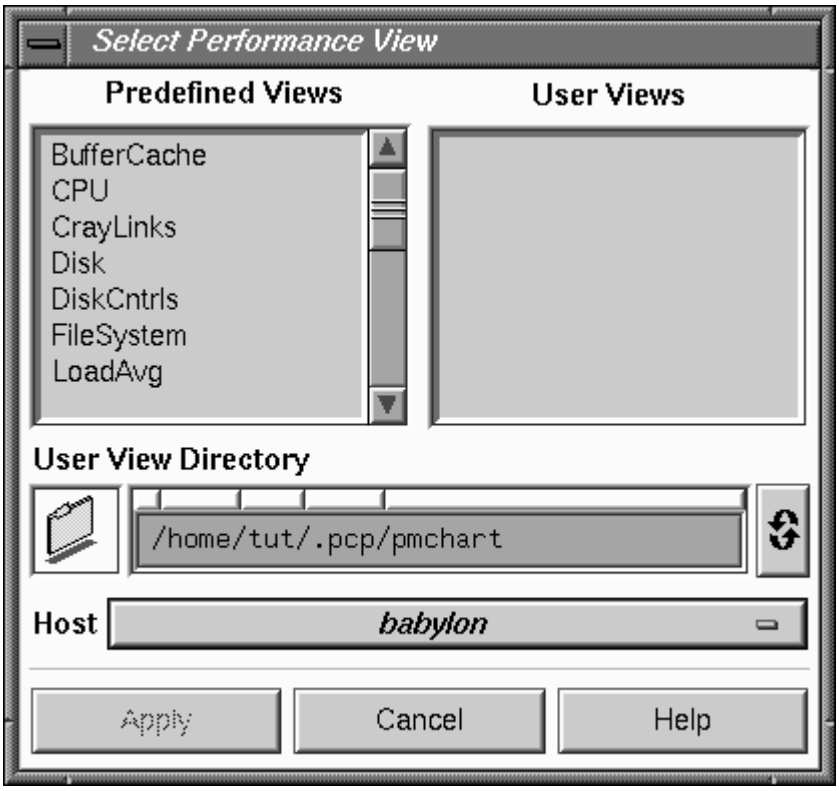

Figure 11. `pmchart Select Performance View` Dialog

You may use this dialog to select one of the available views. The default PCP views include the following:

`BufferCache`	Cumulative amount of data read and written between system buffers and user memory or block devices.
`CPU`	Processor utilization (user, system, memory break, interrupt, I/O wait, and idle time) aggregated over all CPUs.
`CrayLinks`	Usage of CrayLink node connectors, if this hardware is present.
`Disk`	Cumulative number of read and write transfers for all disk devices.
`DiskCntrls`	Cumulative number of read and write transfers for all drives attached to each disk controller on the system.
`FileSystem`	Percentage of each filesystem in use (percent full).
`LoadAvg`	System load averaged over intervals of 1, 5, and 15 minutes.
`Memory`	Memory used by the kernel, filesystems, user processes, and free space.
`NetBytes`	Network interface activity—octets transmitted on various interfaces.
`NetConnDrop`	TCP drops, connection drops, timeout drops, and TCP accepts.
`NetPackets`	Rate of TCP and UDP packets received and sent.
`NetTCPcongestion`	TCP packets retransmitted, retransmit timeouts, and TCP packets sent.
`NFS`	Client and server NFS operation rates.
`Overview`	Composite charts of CPU, LoadAvg, Memory, Disk, and NetBytes.
`Paging`	Page-in and page-out rates from the virtual memory subsystem.
`PMCD`	Message rates and CPU time used by `pmcd` or associated PMDAs.

`Swap`	System swap space allocated, reserved, and unused.
`Syscalls`	Rate of `exec`, `fork`, `read`, `write`, and total system calls.

You can create your own custom views using the metric selection facilities, and save your views for later using the `Save View...` option of the `File` menu.

4.1.3 Displaying Horizontal Lines

You can have `pmchart` display horizontal lines, usually in a lighter background color, at major tick marks by calling `pmchart` with the following arguments (quotes required):

```
% pmchart -xrm "PmChart*xrtYGridUseDefault: True"
```

For greater convenience, you can place the following line in your `$HOME/.Xresources` file, to have `pmchart` always display horizontal lines:

```
PmChart*xrtYGridUseDefault: True
```

4.1.4 `pmchart` Metric Selection

The `pmchart` `Metric Selection` window, shown in Figure 12, allows interactive navigation of the Performance Metrics Name Space (PMNS) to create new chart configurations.

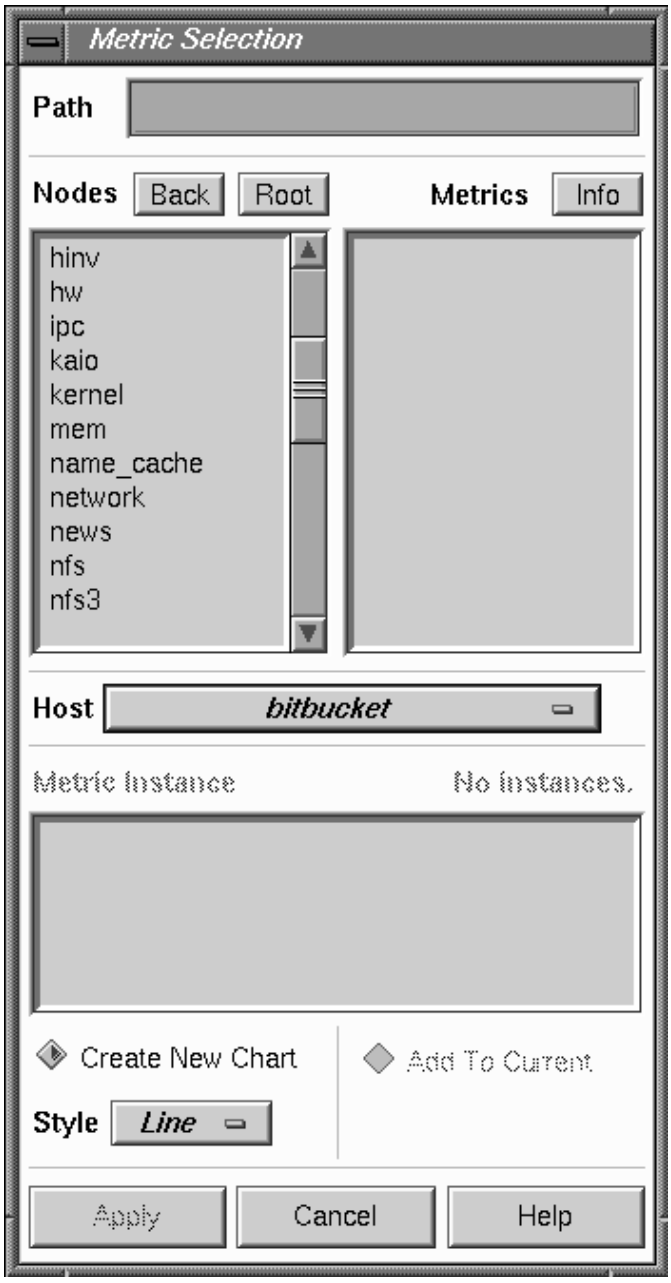

Figure 12. `pmchart Metric Selection` Dialog

You can choose metrics, display information about metrics, change the current host or archive, select metric instances, and plot metric values on a common time axis. You bring up this window by choosing `New Plot...` from the `File` menu of `pmchart`.

Metric selection proceeds by navigating through the tree-structured PMNS. If you enter a partial metric specification in the `Path` field in the `Metric Selection` dialog, you can avoid having to navigate through the PMNS for the metrics you need. For example, if you enter `network.interface`, the window changes dynamically, as shown in Figure 13.

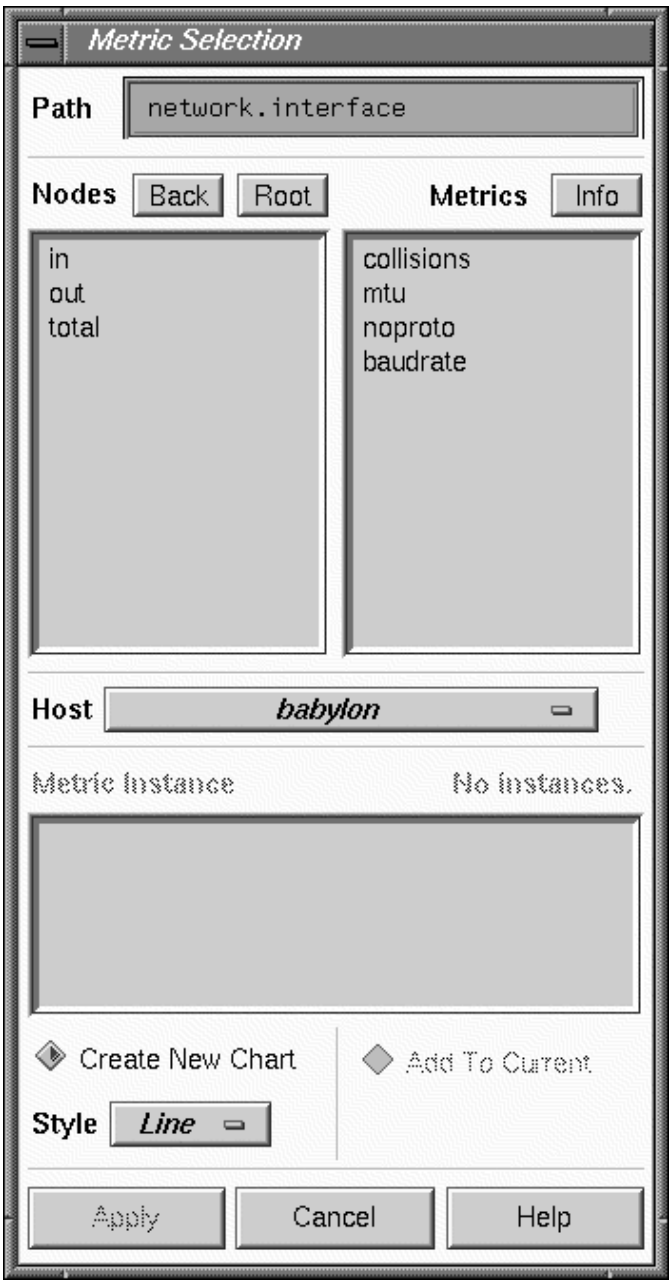

Figure 13. Further `Metric Selection`

You can continue the selection process by choosing non-leaf nodes from the `Nodes` list, and finally a leaf node from the `Metrics` list. At this stage, the `Path` corresponds to a leaf node in the PMNS, as shown in Figure 14.

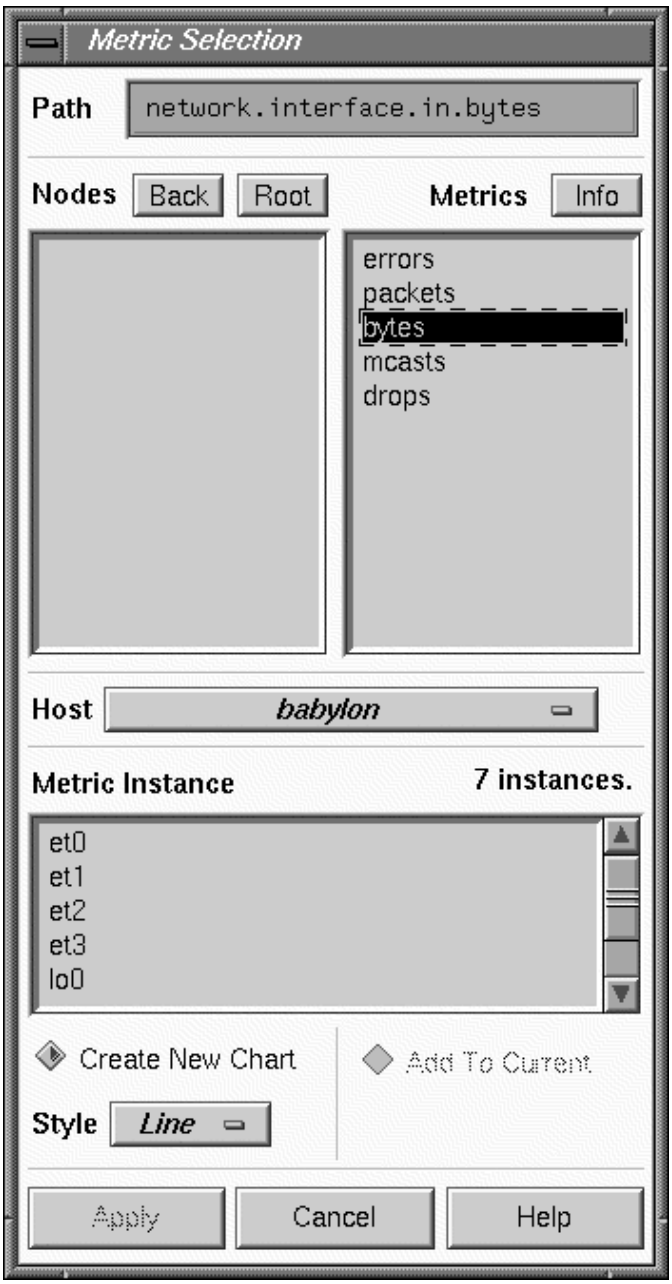

Figure 14. Selecting a Leaf Node in the PMNS (Performance Metric)

Once a metric has been selected, the `Info` button in the `Metric Selection` dialog launches the `Metric Information` dialog, as shown in Figure 15.

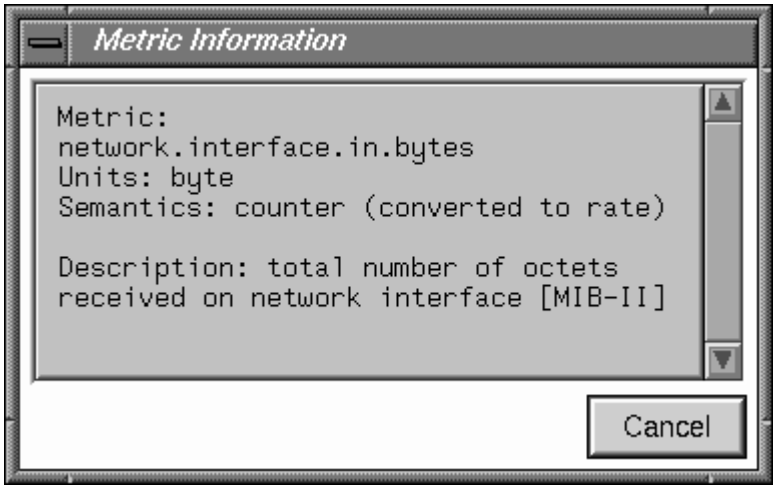

Figure 15. `Metric Information` Dialog

This dialog displays the name, unit, and semantics for the currently selected metric, along with the verbose help text that describes the metric, and optionally a description of the underlying instance domain.

Finally, you may have to select from several instances of a metric. In the example shown in Figure 15, you wish to monitor the input packet rate for some network interface(s). For the current source of performance metrics, there are two network interfaces configured. You must select one or more instances, as shown in Figure 16.

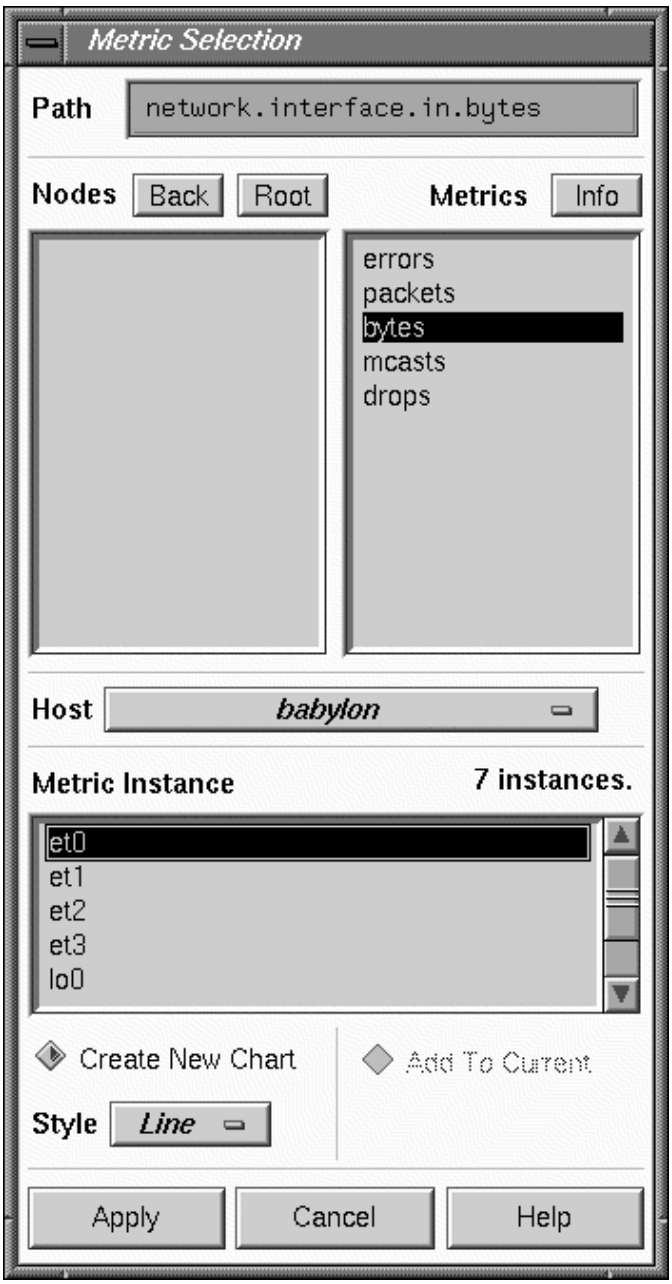

Figure 16. Selecting a Metric Instance

You can select multiple instances either by clicking and dragging up and down the list with the left mouse button, or by selecting the first instance and then using the `Shift` key (or `Ctrl` key) with the left mouse button to select one or more other instances.

4.1.5 Creating a PCP Archive from a `pmchart` Session

From the `File` menu of `pmchart` when running in live mode, the `Record` (`Stop Recording`) option may be used to start (or stop) the creation of a PCP archive log. The archive log is created using `pmlogger` and includes the update interval and all of the performance metrics in the current `pmchart` configuration when recording begins.

> **Note:** Any changes made to the `pmchart` configuration after recording has been started will not be reflected in the archive log. For these to take effect, the recording must be stopped and restarted (thereby creating a second PCP archive log).

When recording is started, a `File Chooser` dialog is launched, and the user must provide the name of a new file to be used as the PCP archive folio for the new archive (see Section 7.4.1). The recording session produces multiple files in the same directory as the archive folio.

If necessary, `pmchart` creates directories on the path to the named archive folio.

It is often convenient to maintain one directory for each new folio, or else one directory for each group of folios related by collector host(s), service type, or chart selection.

When recording is active, a small red indicator appears in the time control button, as shown at the bottom left of Figure 17.

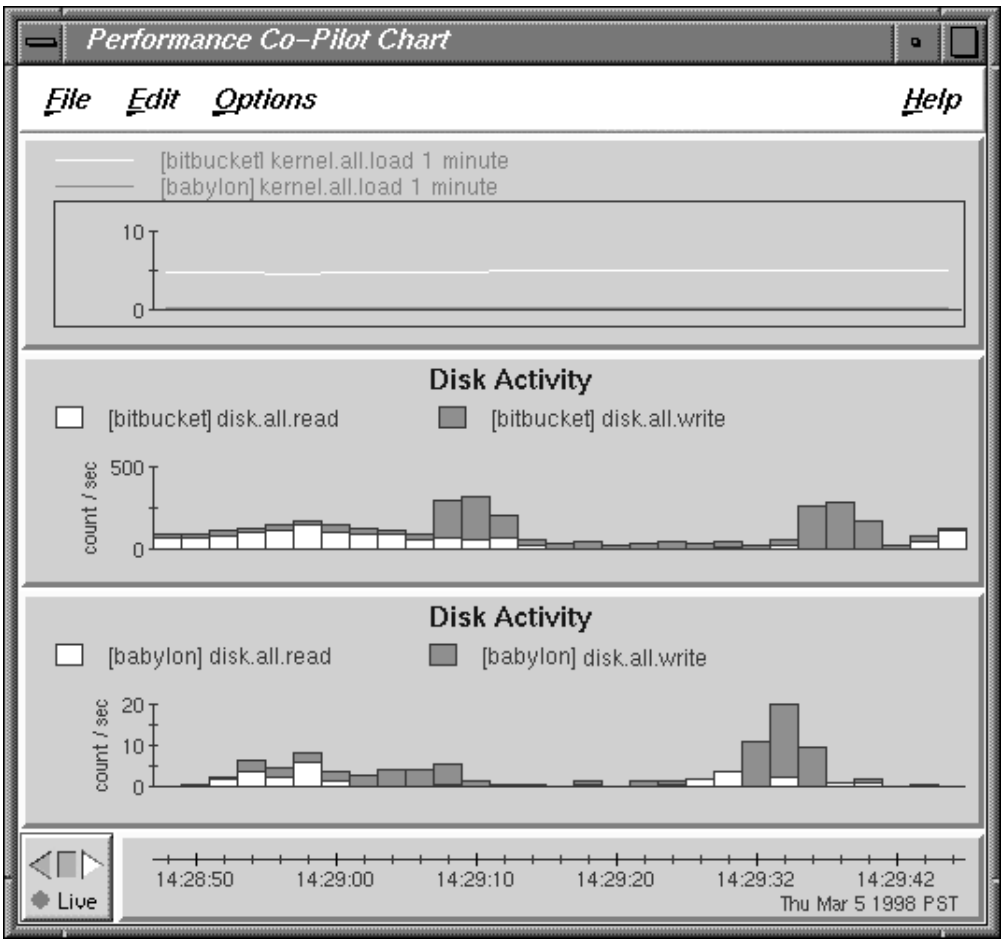

Figure 17. `pmchart` Display When Recording

If you choose `File > Stop Recording`, logging stops immediately. The red light in the lower left turns gray.

To start recording again, chose `File > Record` and specify a new archive folio name.

If you exit `pmchart` by choosing `File > Quit`, an `Archive Recording Session-pmchart` dialog similar to that shown in Figure 18 appears to remind you where the archive folio was created, and to confirm that recording should be terminated.

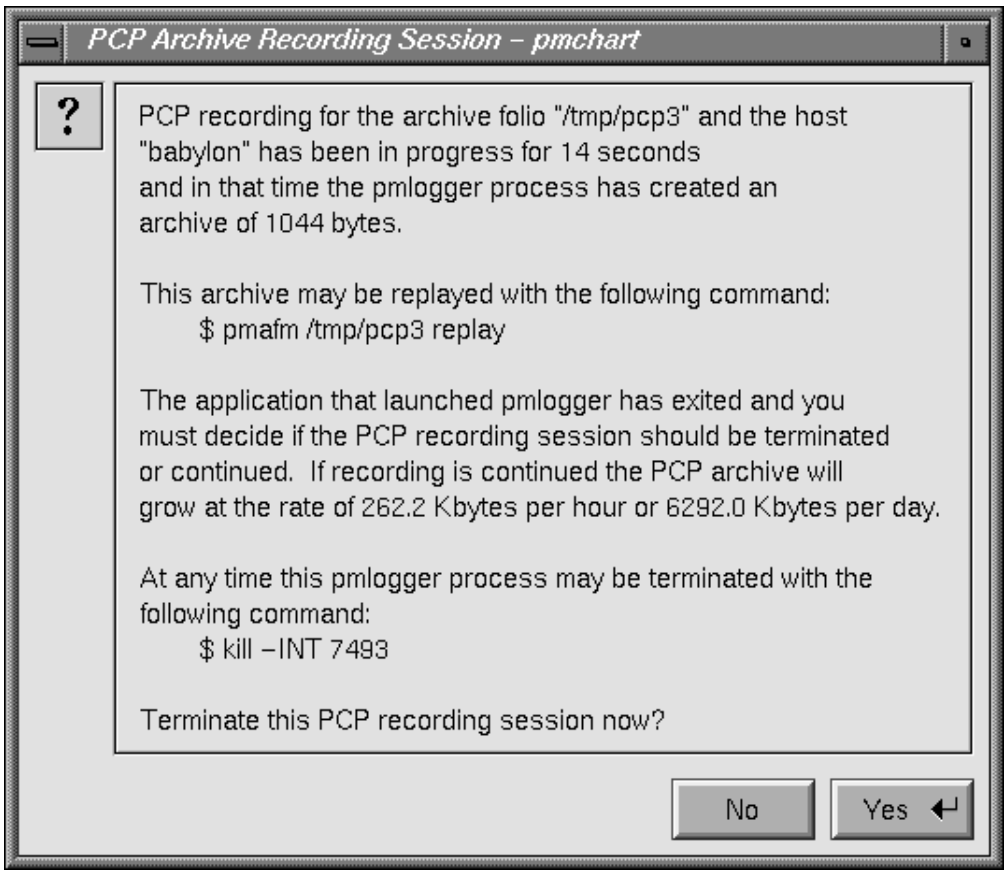

Figure 18. `Archive Recording Session-pmchart` Dialog

If you select `Yes`, recording stops immediately.

If you select `No`, recording continues. This is a useful way to continue archive logging without keeping `pmchart` active.

4.1.6 Changing `pmchart` Colors

When using a video projector, or when making presentations to a large group, or as a result of personal preference, the default pastel color scheme used by `pmchart` may be inappropriate.

The `Colors` option in the `Edit` menu allows arbitrary changes to the colors of individual charts. For more global changes, you can override the defaults using the X11 resources that `pmchart` honors.

For example, create or add the following entries in the `$HOME/.xrdb` file:

```
PmChart*xrtForegroundColor: "green"
PmChart*xrtBackgroundColor: "black"
PmChart*xrtGraphForegroundColor: "rgb:00/b0/00"
PmChart*xrtGraphBackgroundColor: "black"
PmChart*xrtHeaderForegroundColor: "green"
PmChart*xrtHeaderBackgroundColor: "black"
PmChart*pmDefaultColors: rgb:ff/ff/00 rgb:00/ff/00 rgb:00/00/ff \
                           rgb:ff/ff/00 rgb:00/ff/ff rgb:ff/00/ff
```

Use the following command to change the default color scheme for `pmchart` to one with bright primary colors on a black background:

```
xrdb -merge $HOME/.xrdb
```

4.1.7 Other Chart Customizations

The `pmchart Edit` menu provides options and a dialog that you may use to change and customize the display as follows:

`Chart Style`	Choose from line, bar, stacked bar, area plot, and utilization
`Chart Title and Legend...`	Change the chart title, and enable or disable the legend annotation at the top of each chart
`Y-Axis Scaling...`	Fix the maximum and minimum values of the range on the Y-axis, or allow `pmchart` to adjust the range dynamically to reflect currently displayed values
`Colors...`	Customize plot colors
`Delete`	Select all charts, a complete chart, or individual plots from a chart

The `pmchart Options` menu provides another option for customizing the display:

`Visible Points...`	Use the slider to change the number of values along the time axis

4.1.8 Time Control

The `Options` menu provides access to the `PCP Time Control` Dialog (as described in the Section 3.4, page 47).

`Show Time Control`	Exposes the dialog for the controlling `pmtime` instance, thereby allowing users to change the sampling interval. Selecting the `Time Control` button in the lower left corner of the main `pmchart` window also exposes the `Time Control` dialog. If the current source of performance metrics is one or more PCP archive logs, this same dialog may be used for temporal navigation within the archive(s).
`New Time Control`	Detaches `pmchart` from the controlling `pmtime` instance and launches a new `pmtime` instance, initially dedicated to this `pmchart`.
`Launch New Pmchart`	Starts a new `pmchart`, with shared `pmtime` control.

4.1.9 Taking Snapshots of `pmchart` Displays and Value Dialogs

The `Print` option in the `File` menu enables the current `pmchart` display to be printed in a variety of PostScript styles. The output can be saved in a file or sent directly to a printer.

The `-o` option for `pmchart` also provides the facility to produce Graphics Interchange Format (GIF) image snapshots of the `pmchart` display.

It is often convenient to publish performance summary information for the users of a particular computing environment. The `pmchart` tool, in combination with the `pmsnap` script and its associated control files, can be used to produce high-quality performance summary snapshot images in GIF format. These images can be incorporated into Web pages, reports, e-mail, or presentation material.

The following files and utilities are included in support of this feature:

`/var/pcp/config/pmsnap/Summary`

This file contains a summary of the performance metrics used in the example snapshot.

`/var/pcp/config/pmsnap/Summary.html`

An example HTML page suitable for publishing images from the Summary `pmsnap` example via a Web server.

`/var/pcp/config/pmsnap/control`

This file controls the snapshot parameters.

`/var/pcp/config/pmlogger/config.Summary`

This configuration file specifies an archive log suitable for use with any `pmview` -type tool, and the example Summary snapshot configuration.

`/usr/pcp/bin/pmsnap`

The `pmsnap` script is designed to be periodically run by the `cron` command to process the control file `/var/pcp/config/pmsnap/control` and generate snapshot images according to the specifications therein. The `pmsnap`(1) man page describes the command line options for selecting the control lines to process, the default directory for the output files, the `X` display to use, and other parameters.

Instructions for configuring `pmsnap` are in the man page. There is also a verbose comment at the head of the control file. The `pmchart`(1) man page is also useful.

4.1.10 More Information

The annotated examples in the `pmchart` chapter of the PCP Tutorial provide a guided illustration to a typical user's interactions with `pmchart`. The PCP Tutorial can be optionally installed as the `pcp.man.tutorial` subsystem. To view the `pmchart` chapter of the tutorial, open the following URL with your Web browser:

`file:/var/pcp/Tutorial/pmchart.html`

4.2 The pmgadgets Command

The pmgadgets tool creates a small window containing a collection of graphical gadgets driven by performance metrics supplied by the PCP framework. Any numeric metric supported by PCP can be displayed.

> **Note:** In the current PCP release, pmgadgets is constrained to process performance metrics from real-time sources (and not PCP archive logs), although metrics from several different hosts may be displayed simultaneously in the same window.

The layout of the gadgets and the performance metrics that lie behind them are specified in a configuration file, and pmgadgets is typically run on an existing configuration file or in conjunction with an application that automatically generates a configuration file. For example, pmgsys generates a configuration file for various IRIX performance metrics and feeds it to pmgadgets. The resulting display depends on the host configuration, but the display shown in Figure 19 is representative of a system with four CPUs, eleven disks on three controllers, and four network interfaces.

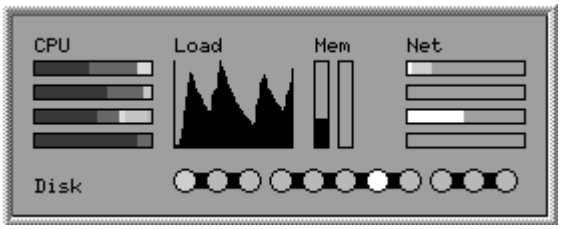

Figure 19. Representative pmgadgets Display Using pmgsys

Other pmgadget front end tools such as pmgcluster, pmgevctr, pmgcisco, and pmgshping are not described in this chapter. For information about these tools, see the pmgcluster(1), pmgevctr(1), pmgcisco(1), and pmgshping(1) man pages

The pmgadgets tool displays much the same information as pmchart, but more compactly, and without so many historical graphs.

The pmgadgets specification language provides the ability to define the following gadgets and components:

_bar — Displays a single performance metric value as a rectangle. This rectangle is filled from left to right

	or bottom to top in proportion to the ratio of the metric's value to some maximum.
`_multibar`	Is similar to the bar gadget, but displays several performance metrics at the same time (same as stacked bar). Each is allocated a color and the gadget's rectangle is filled with an amount of each color proportional to the ratio of the corresponding performance metric's contribution to a maximum value.
`_bargraph`	Displays a simple `xload` style strip chart of a performance metric's values over time.
`_led`	Is a circular gadget whose color is modulated using the value of a single performance metric.
`_line`	Is a solid rectangle, not modulated by any performance metric, useful for highlighting connectivity between gadgets.
`_label`	Provides textual annotation in the display.
`_actions`	Provides customized menus of "drill-down" actions that may be associated with any gadget. Using the right mouse button over a visible gadget causes any associated action menu to pop up.
`_colorlist`	Provides a list of X11 color specifications.
`_legend`	Provides the association between color and range of performance metric values for use in a `_led` gadget.

Each visible gadget must be assigned a Cartesian position in the `pmgadgets` display.

By way of an example, the `pmgadgets` specification shown in Example 2 includes CPU, disk, and load average information from two hosts, and produces a customized `pmgadgets` display like the one shown in Figure 20.

Example 2: Specification File for `pmgadgets`

```
_colourlist cpuColours (blue3 red3 yellow3 cyan3 green3)
_legend diskLegend (
       _default  green3
       15        yellow
```

```
       40        orange
       75        red
)
# host moomba
_label 70 12 "moomba"
_multibar 5 5 30 6
       _metrics (
           moomba:kernel.all.cpu.user
           moomba:kernel.all.cpu.sys
           moomba:kernel.all.cpu.intr
           moomba:kernel.all.cpu.wait.total
           moomba:kernel.all.cpu.idle
       )
       _maximum 0.0
       _colourlist cpuColours
_bargraph 40 5 25 20
       _metric moomba:kernel.all.load["1 minute"]
       _max 1.0
_led 12 16 6 6
       _metric moomba:disk.all.read _legend diskLegend
_led 25 16 6 6
       _metric moomba:disk.all.write _legend diskLegend
# host gonzo
_label 70 39 "gonzo"
_multibar 5 32 30 6
       _metrics (
           gonzo:kernel.all.cpu.user
           gonzo:kernel.all.cpu.sys
           gonzo:kernel.all.cpu.intr
           gonzo:kernel.all.cpu.wait.total
           gonzo:kernel.all.cpu.idle
       )
       _maximum 0.0
       _colourlist cpuColours
_bargraph 40 32 25 20
       _metric gonzo:kernel.all.load["1 minute"]
       _max 1.0
_led 12 43 6 6
       _metric gonzo:disk.all.read _legend diskLegend
_led 25 43 6 6
       _metric gonzo:disk.all.write _legend diskLegend
```

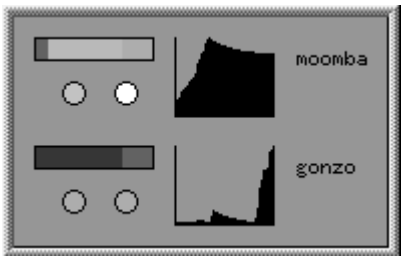

Figure 20. Customized `pmgadgets` Display

In addition to the drill-down capabilities of `pmgadgets`, positioning the cursor over a gadget and entering a space character causes an information dialog to be exposed. This dialog tracks the current values of the performance metrics that are associated with the gadget as illustrated by the `pmgadgets` information dialog in Figure 21.

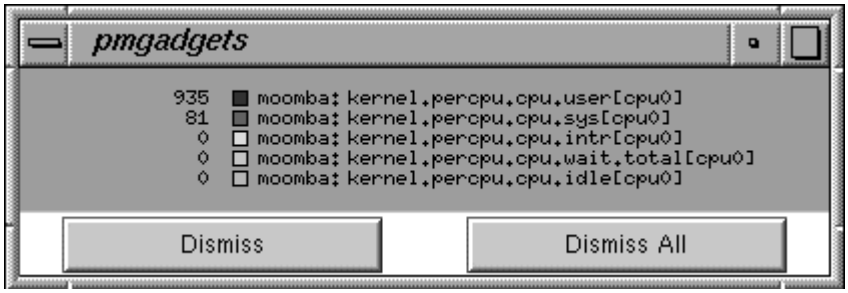

Figure 21. `pmgadgets` Dialog

The `pmgadgets`(1) man page provides a complete description of the gadget specification language and the user interface controls of `pmgadgets`.

4.3 The `pmkstat` Command

The `pmkstat` command provides a periodic, one-line summary of system performance. This command is intended to monitor system performance at the highest level, after which other tools may be used for examining subsystems to observe potential performance problems in greater detail. After entering the

`pmkstat` command, you see output similar to the following, with successive lines appearing periodically:

```
pmkstat
# hostname load avg: 0.26, interval: 5 sec, Thu Jan 19 12:30:13 1995
runq    | memory    |      system      | disks|   cpu
mem swp | free page | scall ctxsw  intr| rd wr|usr sys idl wt
0   0     16268 0     64    19     2396  0  0  0   1   99  0
0   0     16264 0     142   45     2605  0  8  0   2   97  0
0   0     16268 0     308   62     2532  0  1  1   1   98  0
0   0     16268 0     423   88     2643  0  0  1   1   97  0
```

An additional line of output is added every five seconds. The update interval may be varied using the `-t` *interval* option.

The output from `pmkstat` is directed to standard output, and the columns in the report are interpreted as follows:

`runq`	Average number of runnable processes in main memory (`mem`) and in swap memory (`swp`) during the interval.
`memory`	The free column indicates average `free` memory during the interval, in kilobytes. The `page` column is the average number of page-out operations per second during the interval. I/O operations caused by these page-out operations are included in the disk write I/O rate.
`system`	System call rate (`scall`), context switch rate (`ctxsw`), and interrupt rate (`intr`). Rates are expressed as average operations per second during the interval.
`disks`	Aggregated physical read (`rd`) and write (`wr`) rates over all disks, expressed as physical I/O operations issued per second during the interval. These rates are independent of the I/O block size.
`cpu`	Percentage of CPU time spent executing user code (`usr`), system and interrupt code (`sys`), idle loop (`idl`) and idle waiting for resources (`wt`), typically disk I/O.

As with most PCP utilities, real-time metric, and archive logs are interchangeable.

For example, the following command uses the PCP archive log *foo* and the time zone of the host (`tokyo`) from which performance metrics in the archive were collected:

```
pmkstat -a foo -z
Note: timezone set to local timezone of host "tokyo"
# tokyo load avg: 1.06, interval: 5 sec, Thu Feb  2 08:42:55 1995
 runq  |     memory |     system       |  disks  |      cpu
mem swp|   free page| scall ctxsw  intr|  rd   wr|usr sys idl  wt
  0   0    4316    0    195    64  2242   32   21   0   3   8  89
  0   0    3976    0    279    86  2143   50   17   0   5   8  87
  1   0    3448    0    186    63  2304   35   14   0   4   9  87
  0   0    4364    0    254    81  2385   35    0   0   4   9  87
  0   0    3696    0    266    92  2374   41    0   0   3   9  88
  0   0    2668   42    237    81  2400   44    2   1   4   7  89
  0   0    4644  100    206    68  2590   25    1   0   3   5  91
  0   0    5384    0    174    63  2296   32   22   0   2   8  89
  0   0    4736    0    189    65  2197   31   28   0   3   8  89
pmFetch: End of PCP archive log
```

For complete information on `pmkstat` usage and command line options, see the `pmkstat`(1) man page.

4.4 The `pmdumptext` Command

The `pmdumptext` command displays performance metrics in ASCII tables, suitable for export into databases or report generators. It is a flexible command. For example, the following command provides continuous memory statistics on a host named *serv*:

```
pmdumptext -imu -h serv -f '%H:%M:%S' mem.util
Metric        kernel  fs_ctl  _dirty  _clean    free    user
      Units            b       b       b       b       b       b
20:14:28        99.14M   6.03M   0.85M  98.42M   0.17G   0.16G
```

See the `pmdumptext`(1) man page for more information.

4.5 The `pmval` Command

The `pmval` command dumps the current values for the named performance metrics. For example, the following command reports the value of performance

metric `proc.nprocs` once per second (by default), and produces output similar to this:

```
pmval proc.nprocs
metric:    proc.nprocs
host:      localhost
semantics: instantaneous value
units:     none
samples:   indefinite
interval:  1.00 sec
            73
            72
            70
            75
            75
```

In this example, the number of running processes was reported once per second.

Where the semantics of the underlying performance metrics indicate that it would be sensible, `pmval` reports the rate of change or resource utilization.

For example, the following command reports idle processor utilization for each of four CPUs on the remote host `moomba`, each five seconds apart, producing output of this form:

```
pmval -h moomba -t 5sec -s 4 kernel.percpu.cpu.idle
metric:    kernel.percpu.cpu.idle
host:      moomba
semantics: cumulative counter (converting to rate)
units:     millisec (converting to time utilization)
samples:   4
interval:  5.00 sec
        cpu0        cpu1        cpu2        cpu3
        0.8193      0.7933      0.4587      0.8193
        0.7203      0.5822      0.8563      0.7303
        0.6100      0.6360      0.7820      0.7960
        0.8276      0.7037      0.6357      0.6997
```

Similarly, the following command reports disk I/O read rate every minute for just the disk `/dev/dsk/dks0d1`, and produces output similar to the following:

```
pmval -t 1min -i dks0d1 disk.dev.read
metric:    disk.dev.read
host:      localhost
semantics: cumulative counter (converting to rate)
```

```
units:     count (converting to count / sec)
samples:   indefinite
interval:  60.00 sec
        dks0d1
         33.67
         48.71
         52.33
         11.33
         2.333
```

The `-r` flag may be used to suppress the rate calculation (for metrics with counter semantics) and display the raw values of the metrics.

When used in conjunction with a PCP archive, the `-g` option may be used to associate a PCP time control dialog (see Section 3.4) with the execution of `pmval` to support temporal navigation within the archive. In the example below, manipulation of the time within the archive is achieved by the exchange of time control messages between `pmval` and `pmtime`.

```
pmval -g -a /var/adm/pcplog/myserver/960801
```

The `pmval` command is documented by the `pmval`(1) man page, and annotated examples of the use of `pmval` are in the PCP Tutorial.

4.6 The `pmem` Command

The `pmem` command reports per-process memory usage statistics within the PCP framework.

Both virtual size and prorated physical memory usage are reported. The virtual memory usage statistics represent the total virtual size of each process, irrespective of how many pages are valid (resident). Prorated physical memory statistics indicate real memory usage (only valid pages are counted) and are prorated on a per-page basis between all processes that reference each page. Thus the prorated physical memory counts reflect the real memory demands for individual processes in the context of the current process mix.

The output of `pmem` can be very large. Here is an abbreviated example of `pmem` output:

```
Host: gonzo Configured: 65536  Free:18380  Tue Jul 9 16:45:08 1996
     pid  ppid   user  vtxt  ptxt  vdat  pdat vshm pshm command
       1     0   root   232   144    84    76    0    0 /etc/init
     832   827   root  3204  1013  5796  3096    0    0 /usr/bin/X11/Xsg
```

```
      221     1   root  1424    54   156   84     0    0 /usr/lib/saf/sad
      838   827   root  2948    36   268    75    0    0 /usr/bin/X11/xdm
       86     1   root  1264    32   144    76    0    0 /usr/etc/syslogd
      182     1   root  1476   129   596   387    0    0 /usr/etc/rpcbind
      827     1   root  2948    13   252    22    0    0 /usr/bin/X11/xdm
      172     1   root  1276    52   148   100    0    0 /usr/etc/routed
Total     vtxt  ptxt   vdat  pdat  vshm  pshm  77 user processes
             121M        36256           0        =  157M virtual
                   13982        20194           0  = 34176 physical
```

The columns report the following information:

`pid`	Process ID number.
`ppid`	Parent process ID number.
`user`	Login name of the process owner.
`vtxt`	Total virtual memory used by text (executable code) regions mapped by the process.
`ptxt`	Prorated physical memory used by text regions.
`vdat`	Total virtual memory used by all non-executable regions, excluding shared memory regions. This includes initialized data, bss, and stack but not shared memory regions.
`pdat`	Prorated physical memory used by all data regions (data, bss, and stack but not shared memory regions).
`vshm`	Total virtual memory used by all shared memory regions.
`pshm`	Prorated physical memory used by shared memory regions.
`command`	The command and arguments.

For complete information on `pmem` usage and command line options, see the `pmem`(1) man page.

4.7 The `pminfo` Command

The `pminfo` command displays various types of information about performance metrics available through the Performance Co-Pilot (PCP) facilities.

The `-T` option is extremely useful; it provides help text about performance metrics:

```
pminfo -T mem.util.fs_dirty
mem.util.fs_dirty
Help:
The amount of memory in Kbytes that is holding file system data.
```

The `-t` option displays the one-line help text associated with the selected metrics. The `-T` option prints more verbose help text.

Without any options, `pminfo` verifies that the specified metrics exist in the name space, and echoes those names. Metrics may be specified as arguments to `pminfo` using their full metric names. For example, this command returns the following response:

```
pminfo hinv.ncpu network.interface.total.bytes
hinv.ncpu
network.interface.total.bytes
```

A group of related metrics in the name space may also be specified. For example, to list all of the `hinv` metrics you would use this command:

```
pminfo hinv
hinv.ncpu
hinv.cpuclock
hinv.dcache
hinv.icache
hinv.secondarycache
hinv.physmem
hinv.pmeminterleave
hinv.ndisk
```

If no metrics are specified, `pminfo` displays the entire collection of metrics. This can be useful for searching for metrics, when only part of the full name is known. For example, this command returns the following response:

```
pminfo | grep nfs
nfs.client.badcalls
nfs.client.badcalls
nfs.client.calls
nfs.client.nclget
nfs.client.nclsleep
nfs.client.reqs
nfs.server.badcalls
```

```
nfs.server.calls
nfs.server.reqs
nfs.client.badcalls
nfs.client.calls
nfs.client.nclget
nfs.client.nclsleep
nfs.client.reqs
nfs.server.badcalls
nfs.server.calls
nfs.server.reqs
```

The `-d` option causes `pminfo` to display descriptive information about metrics (refer to the `pmLookupDesc`(3) man page for an explanation of this metadata information). The following command and response show use of the `-d` option:

```
pminfo -d proc.nprocs disk.dev.read filesys.free
    proc.nprocs
       Data Type: 32-bit int  InDom: PM_INDOM_NULL 0xffffffff
       Semantics: instant  Units: none
disk.dev.read
       Data Type: 32-bit unsigned int  InDom: 1.2 0x400002
       Semantics: counter  Units: count
filesys.free
       Data Type: 32-bit int  InDom: 1.7 0x400007
       Semantics: instant  Units: Kbyte
```

The `-f` option to `pminfo` forces the current value of each named metric to be fetched and printed. In the example below, all metrics in the group *hinv* are selected:

```
pminfo -f hinv
hinv.ncpu
       value 1
hinv.cpuclock
       value 100
hinv.dcache
       value 8192
hinv.icache
       value 8192
hinv.secondarycache
       value 1048576
hinv.physmem
       value 64
hinv.pmeminterleave
```

```
       value 0
hinv.ndisk
       value 1
```

The `-h` option directs `pminfo` to retrieve information from the specified host. If the metric has an instance domain, the value associated with each instance of the metric is printed:

```
pminfo -h babylon.engr.sgi.com -f filesys.mountdir
filesys.mountdir
       inst [1 or "/dev/root"] value "/"
       inst [2 or "/dev/dsk/dks1d3s7"] value "/usr2"
       inst [3 or "/dev/dsk/dks3d1s7"] value "/dbv"
       inst [4 or "/dev/dsk/dks3d4s7"] value "/dbv/d4"
       inst [5 or "/dev/dsk/dks3d2s7"] value "/dbv/d2"
       inst [6 or "/dev/dsk/dks3d3s7"] value "/dbv/d3"
       inst [7 or "/dev/dsk/dks2d4s7"] value "/vicepb"
       inst [8 or "/dev/dsk/xlv/build9"] value "/build9"
       inst [9 or "/dev/dsk/xlv/build8"] value "/build8"
       inst [10 or "/dev/dsk/xlv/lv9.xfs"] value "/lv9"
       inst [11 or "/dev/dsk/dks2d5s7"] value "/usenet"
       inst [12 or "/dev/dsk/xlv/work"] value "/usr/work"
       inst [13 or "/dev/dsk/xlv/build10"] value "/build10"
       inst [14 or "/dev/dsk/xlv/dist"] value "/usr/dist"
       inst [15 or "/dev/dsk/xlv/people"] value "/usr/people"
       inst [16 or "/dev/dsk/xlv/build12"] value "/build12"
       inst [17 or "/dev/dsk/xlv/build11"] value "/build11"
```

The `-m` option prints the Performance Metric Identifiers (PMIDs) of the selected metrics. This is useful for finding out which PMDA supplies the metric. For example, the output below identifies the PMDA supporting domain 4 (the leftmost part of the PMID) as the one supplying information for the metric `environ.extrema.mintemp`:

```
pminfo -m environ.extrema.mintemp
environ.extrema.mintemp PMID: 4.0.3
```

The `-v` option verifies that metric definitions in the name space correspond with supported metrics, and checks that a value is available for the metric. Descriptions and values are fetched, but not printed. Only errors are reported.

Some instance domains are not enumerable. That is, it is not possible to ask for all of the instances at once. Only explicit instances may be fetched from such instance domains. This is because instances in such a domain may have a very short lifetime or the cost of obtaining all of the instances at once is very high.

The *proc* metrics are an example of such an instance domain. The `-f` option is not able to fetch metrics with non-enumerable instance domains; however, the `-F` option tells `pminfo` to obtain a snapshot of all of the currently available instances in the instance domain and then to retrieve a value for each.

Complete information on the `pminfo` command is found in the `pminfo`(1) man page. There are examples of the use of `pminfo` in the PCP Tutorial.

4.8 The `pmstore` Command

From time to time you may wish to change the value of a particular metric. Some metrics are counters that may need to be reset, and some are simply control variables for agents that collect performance metrics. When you need to change the value of a metric for any reason, the command to use is `pmstore`.

> **Note:** For obvious reasons, the ability to arbitrarily change the value of a performance metric is not supported. Rather, the PMCS selectively allows some metrics to be modified in a very controlled fashion.

The basic syntax of the command is as follows:

`pmstore` *metricname value*

There are also command line flags to further specify the action. For example, the `-i` option restricts the change to one or more instances of the performance metric.

The *value* may be in one of several forms, according to the following rules:

1. If the metric has an integer type, then *value* should consist of an optional leading hyphen, followed either by decimal digits or “0x” and some hexadecimal digits; “0X” is also acceptable instead of “0x.”
2. If the metric has a floating point type, then *value* should be in the form of an integer (described above), a fixed point number, or a number in scientific notation.
3. If the metric has a string type, then *value* is interpreted as a literal string of ASCII characters.
4. If the metric has an aggregate type, then an attempt is made to interpret *value* as an integer, a floating point number, or a string. In the first two cases, the minimal word length encoding is used; for example, “123” would be interpreted as a four-byte aggregate, and “0x100000000” would be interpreted as an eight-byte aggregate.

The following example illustrates the use of `pmstore` to enable performance metrics collection in the `txmon` PMDA (see `/usr/pcp/pmdas/txmon` for the source code of the txmon PMDA). When the metric *txmon.control.level* has the value 0, no performance metrics are collected. Values greater than 0 enable progressively more verbose instrumentation.

```
pminfo -f txmon.count
txmon.count
No value(s) available!
pmstore txmon.control.level 1
txmon.control.level old value=0 new value=1
pminfo -f txmon.count
txmon.count
       inst [0 or "ord-entry"] value 23
       inst [1 or "ord-enq"] value 11
       inst [2 or "ord-ship"] value 10
       inst [3 or "part-recv"] value 3
       inst [4 or "part-enq"] value 2
       inst [5 or "part-used"] value 1
       inst [6 or "b-o-m"] value 0
```

For complete information on `pmstore` usage and syntax, see the `pmstore`(1) man page.

System Performance Visualization Tools [5]

Several 3D graphical tools are provided with Performance Co-Pilot to assist you in visualizing performance on monitored systems. These tools are implemented with and require Inventor, a 3D graphics facility. Each tool is completely described by its own man page, accessible through the `man` command. For example, the man page for the `pmview` tool can be viewed by giving the following command:

```
man pmview
```

The following major sections are covered in this chapter:

- Section 5.1, page 95, provides background motivation and places the current chapter in the context of other PCP tools.
- Section 5.2, page 97, describes the graphical disk activity visualization tool, `dkvis`.
- Section 5.3, page 99, describes the graphical multiprocessor visualization and comparison tool, `mpvis`.
- Section 5.4, page 101, describes the operating system activity visualization tool, `osvis`.
- Section 5.5, page 103, describes the Origin visualization tool, `oview`.
- Section 5.6, page 104, describes the graphical NFS activity visualization tool, `nfsvis`.
- Section 5.7, page 107, describes the graphical performance visualization tool, `pmview`, on which the other visualization tools are based.

Other PCP visualization tools, such as `nodevis`, `routervis`, `txmonvis`, and `xbowvis`, are not described in this chapter. See the `nodevis`(1), `routervis`(1), `txmonvis`(1), and `xbowvis`(1) man pages for information about these `pmview` based tools.

5.1 Overview of Visualization Tools

For the most interesting and complex problems in performance management, the volume of available information is daunting. One approach to dealing with the volume and complexity of the information is to employ automated reasoning. Refer to Chapter 6, for a complete description of the `pmie` tool that

provides this capability. Another approach is to harness the considerable potential for human visual processing to absorb, analyze, and classify large amounts of information.

Performance Co-Pilot has been developed with an assumption that being able to draw three-dimension pictures of system performance is a critical monitoring requirement, and one that offers vast potential for increased insight and understanding for the person charged with some aspect of performance monitoring and management.

Building on SGI technologies of high-performance 3D graphics at the workstation, OpenGL and Open Inventor, PCP delivers a range of utilities, services, and toolkits that are designed both to provide basic visualization tools and to foster the local customization of value-added tools to meet the needs of end-user application and operational environments.

Key components to this performance visualization strategy are as follows:

- Time-series strip charts with `pmchart` (described in Chapter 4) that allow performance metrics from multiple hosts and multiple Performance Metric Domains to be concurrently displayed on a single correlated time axis.

  Predefined chart configurations for common performance scenarios are provided.

- Basic three-dimension models for the following:

  – Per-processor CPU utilization with `mpvis`

  – Per-disk spindle activity with `dkvis`

  – NFS request traffic with `nfsvis`

  – Additional scenes provided with the optional PCP add-on products

- A generalized, three-dimension performance model viewer, `pmview`, that can easily be configured to draw scenes animated by the values of arbitrarily selected performance metrics. Tools like `mpvis`, `dkvis`, and `nfsvis` are front-ends that create scene descriptions to be displayed and animated with `pmview`.

- Icon-sized stripcharts, meters, and indicator LEDs that may be combined into a desktop control and indicator panel using `pmgadgets`; see Section 4.2.

  The `pmgsys` command provides a standard layout for important IRIX performance metrics using `pmgadgets`. The color or height of each gadget

is modulated by user-selected performance metrics from one or more PCP sources.

- `opsview`, a sample multilevel performance visualization for Oracle parallel server configurations, supports drill-down navigation and links several different visualization paradigms.

When combined with the VCR and archive services of Performance Co-Pilot, these visualization tools provide both real-time and retrospective analysis of system performance at many different levels of detail.

5.2 The `dkvis` Disk Visualization Tool

The `dkvis` tool is a graphical disk device utilization viewer, displaying a bar chart showing disk activity. When you give the `dkvis` command, you see a bar chart displaying activity on each disk on the monitored system. You see a `Total Disk I/O Rate for Host` *host* window similar to the one shown in Figure 22.

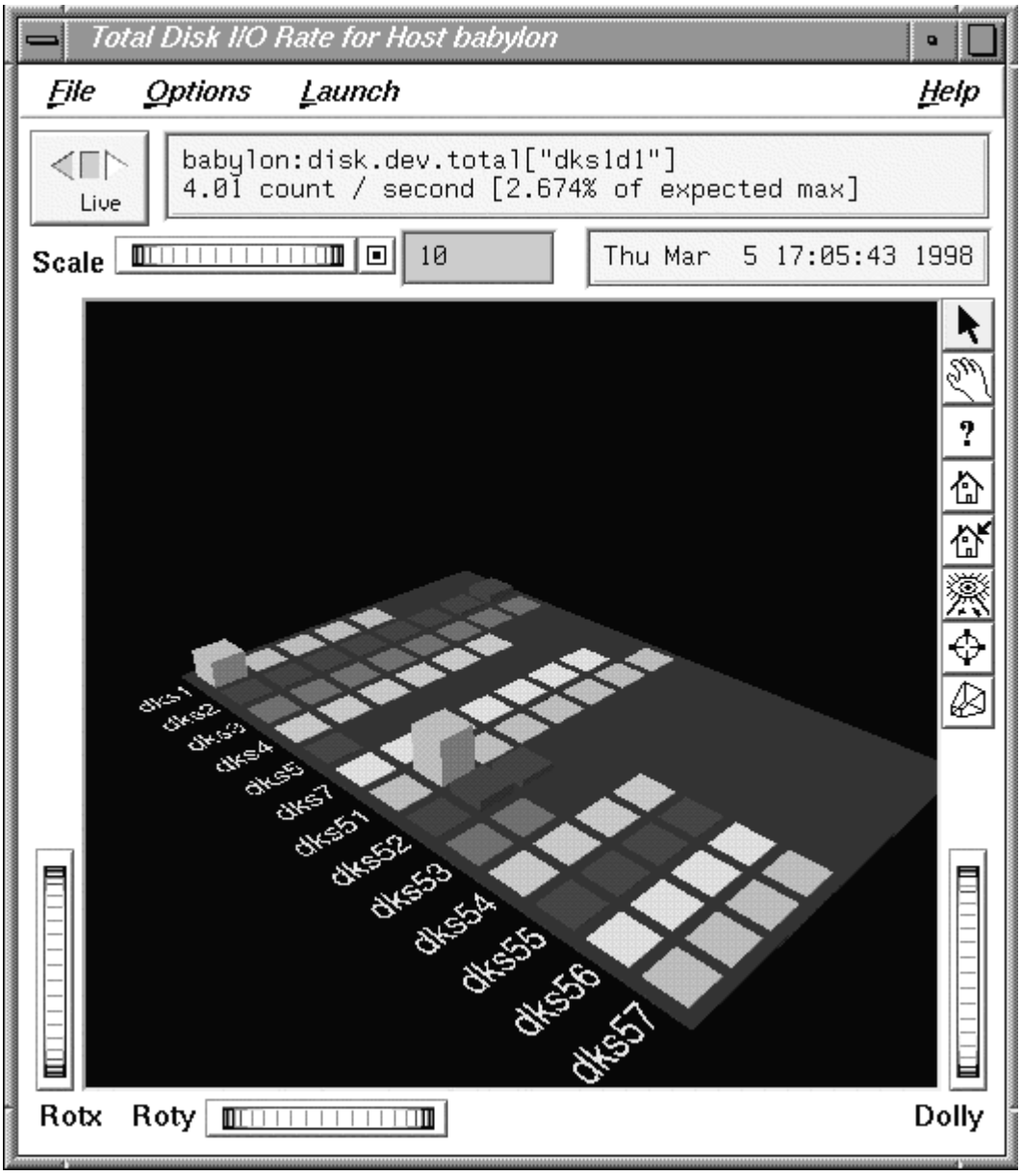

Figure 22. `dkvis Total Disk I/O Rate` Window

Each row of blocks on the base plane represents the group of disks connected to a single disk controller (or host adaptor or SCSI bus). The label for each row is generated from the characters common to the names of all of the disks on the controller. For example, in Figure 22, the disks in the row labeled `dks56` (the same row as the selected block for `dks56d2`) are `dks56d1`, `dks56d2`, `dks56d3`, and `dks56d4`.

The `dkvis` implementation uses the generalized 3D performance viewer `pmview` as described in Section 5.7. Hence, the command line options for `dkvis` include the common ones for `pmview`.

`dkvis` normally displays the total number of I/O operations per second (IOPS). The `-r` option may be used to restrict the display to just the read operations or `-w` may be specified for just the writes.

The `dkvis` command expresses the utilizations in the information window as percentages of some maximum expected rate (clipped to 100%). The `-m` flag allows you to override the default maximum value. This is useful if all of the utilizations are small compared to the maximum. In such a situation, specifying a smaller maximum has the effect of magnifying the differences between the blocks. Similarly, if some of the blocks are almost always at full height, there is a good chance that they are being clipped.

A suitable value for the `-m` option can be determined by clicking the blocks in question, observing the values displayed in the information window for a while, and adding about 10% to the highest value observed. Interactive adjustment of the block height is available via the scale thumb wheel in the `pmview` viewer.

Complete information on the `dkvis` command is available in the `dkvis`(1) man page. The PCP Tutorial contains additional examples on the use of `dkvis`.

5.3 The `mpvis` Processor Visualization Tool

The `mpvis` tool is a graphical multiprocessor activity viewer, displaying a bar chart that shows processor activity. When you enter the `mpvis` command, you see a bar chart displaying activity on each processor on the monitored system. You see a `CPU Utilization for Host` *host* window similar to the one shown in Figure 23.

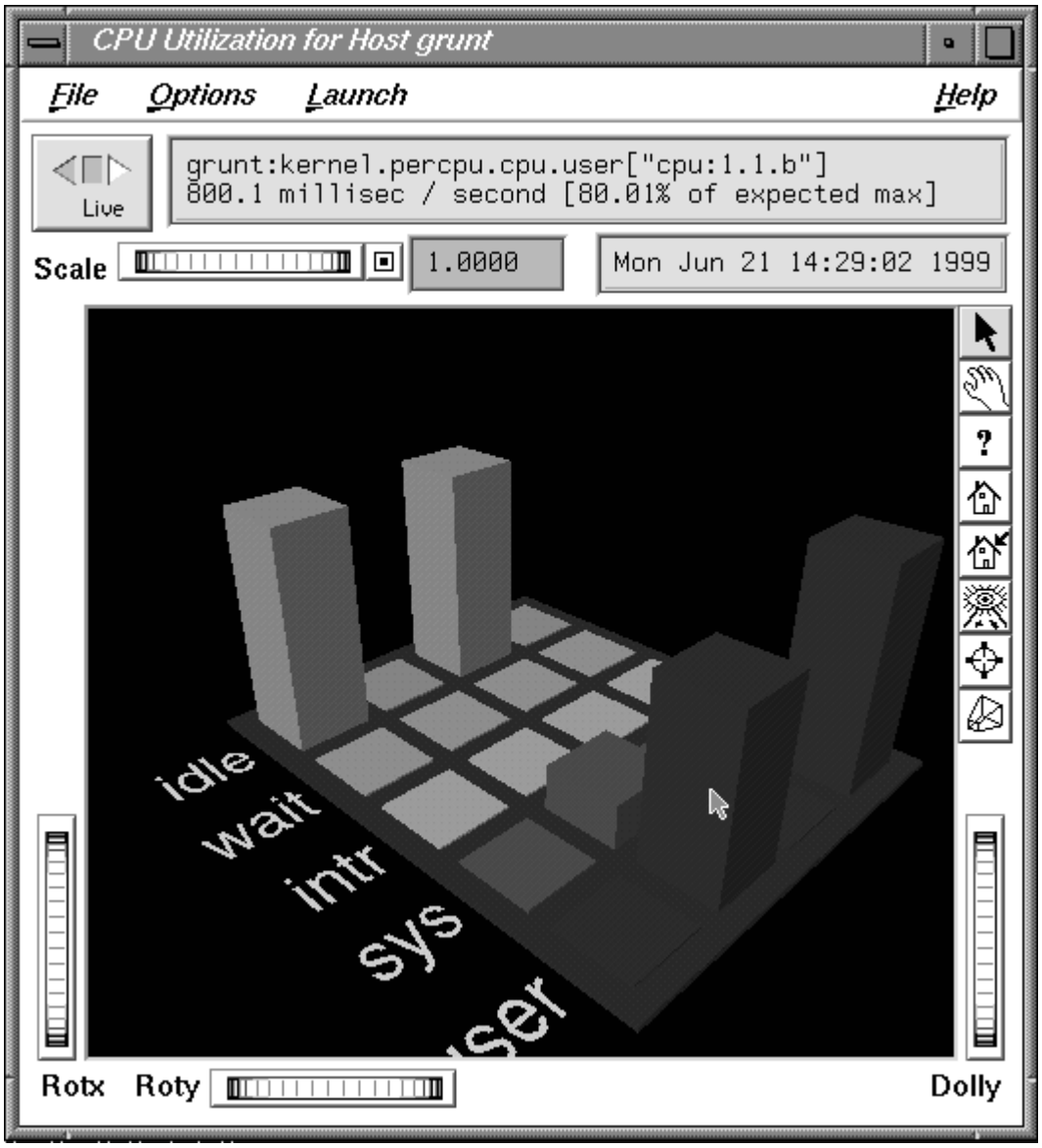

Figure 23. `mpvis CPU Utilization` Window

This figure shows `mpvis` monitoring a machine with four CPUs. CPU is spending 80% of its time processing user code and about 20% of its time

executing system code. Another CPU is executing user code for close to 90% of the time. The remaining two CPUs are idle.

The display contains five labeled rows of blocks, which represent the breakdown of the activity of a single CPU into five states. There is one column of five blocks for each CPU on the system being monitored. These five states are as follows:

`idle`	No activity
`wait`	Like idle but waiting for I/O
`intr`	Processing an interrupt
`sys`	Executing in the IRIX kernel
`user`	Executing user code

The `mpvis` implementation uses the generalized 3D performance viewer `pmview` as described in Section 5.7. Hence, the command line options for `mpvis` include the common ones for `pmview`.

Complete information on the `mpvis` command is available in the `mpvis`(1) man page. The PCP Tutorial contains additional examples on the use of `mpvis`.

5.4 The `osvis` System Visualization Tool

The `osvis` tool displays a high-level overview of performance statistics collected from the PCP infrastructure. The display is modulated by the values of performance metrics retrieved from the target host or from the PCP archive log identified with the `-a` option. Figure 24 shows a sample `osvis` `High-Level Activity for Host` *host* display.

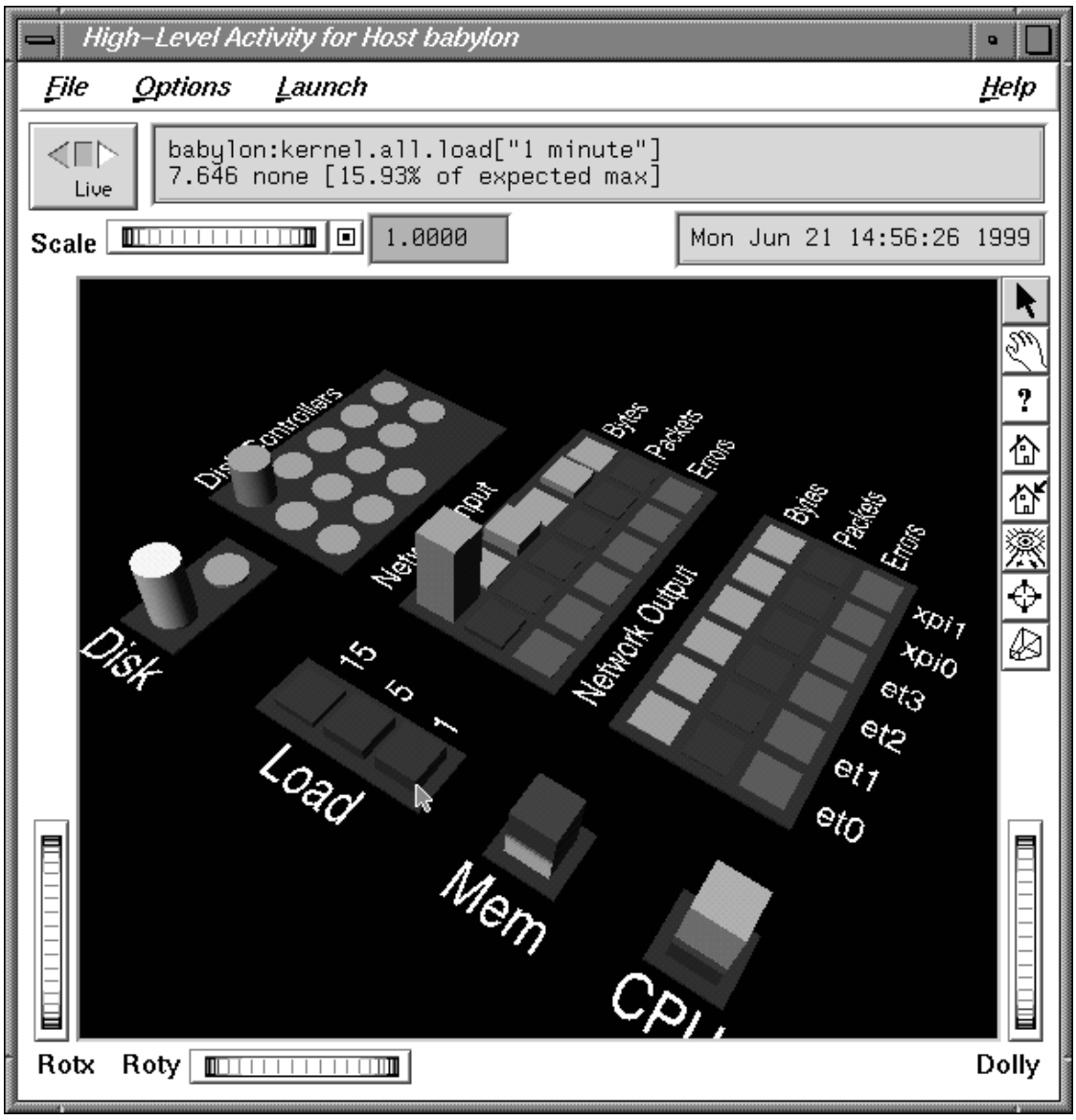

Figure 24. `osvis High-Level Activity` Window

As in all `pmview` scenes, when the mouse is moved over one of the bars, the current value and metric information for that bar are shown in the text box near

the top of the display. The height and color of the bars is proportional to the performance metric values relative to the maximum expected activity.

The bars in the `osvis` scene represent the following information:

`Disk`	The first stack is the rate of disk read and write operations aggregated over all disk spindles. The second bar is the average time the disks are busy, which approximates average time utilization of all disks.
`Load`	The three bars represent average load for the past 1, 5, and 15 minutes, normalized by twice the number of CPUs on the machine.
`Mem`	The stack shows memory utilization by breaking down real memory into kernel, file system, and user usage. The memory utilization metrics (*mem.util*) may not be available on all hosts, so this may only show the amount of free memory as a single bar on some hosts.
`CPU`	This bar shows CPU utilization, aggregated over all CPUs.
`Disk Controllers`	The average time the disks were busy on each disk controller, which approximates the average time utilization of all disks on each controller.
`Network Input`	The first two rows of bars show the input byte rate and the input packet rate for each network interface, except loopback and slip interfaces.
`Network Output`	The first two rows of bars show the output byte rate and the packet rate for each network interface, except loopback and slip interfaces.

5.5 The `oview` Origin Visualization Tool

The `oview` tool displays a dynamic display of Origin system topology and performance, as shown in Figure 25. It displays performance information about CPUs, nodes, and routers in Origin systems connected in various configurations; see the `oview`(1) man page for details.

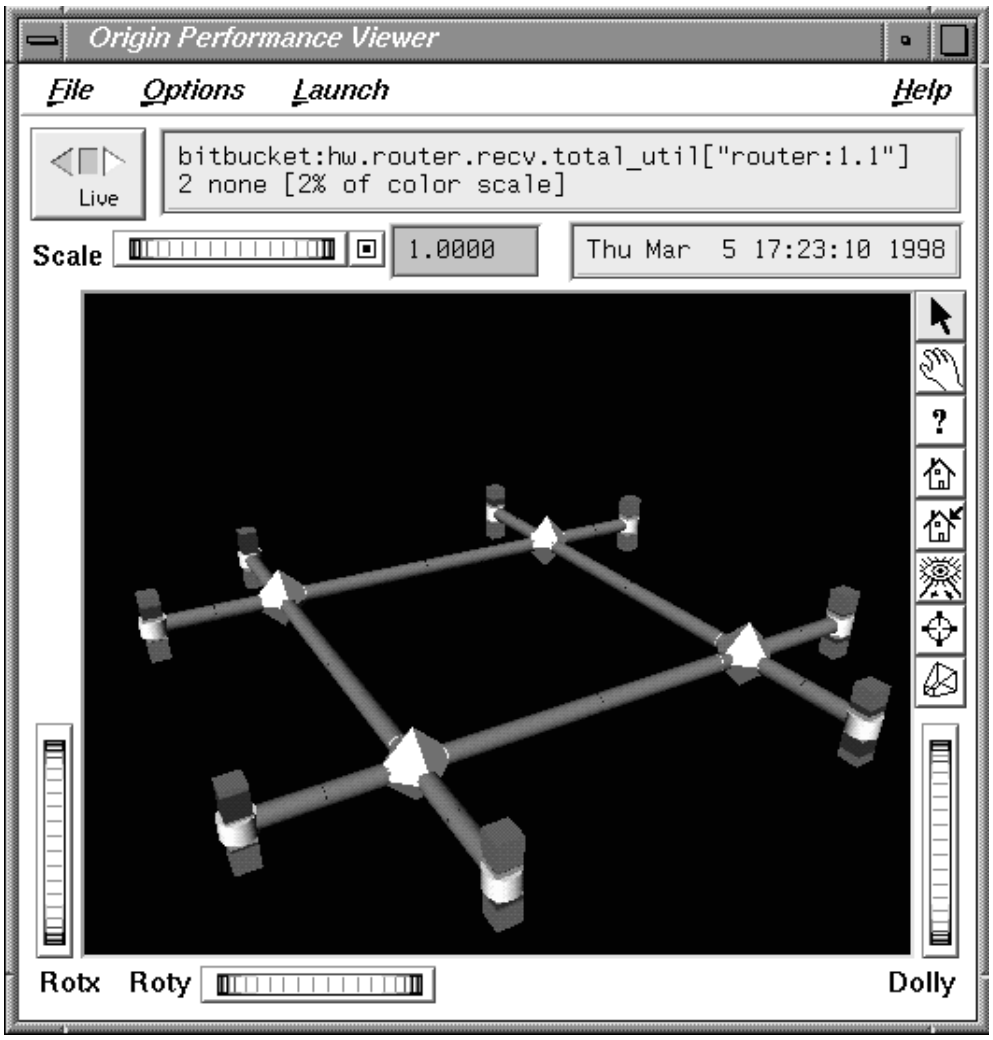

Figure 25. `oview` Window

5.6 The `nfsvis` NFS Activity Visualization Tool

The `nfsvis` tool is a graphical NFS (Network File System) activity viewer, displaying a bar chart that shows NFS request activity on the monitored system. NFS is optional software, and may not be present on all systems or at all sites.

When you run the `nfsvis` command, you see a bar chart displaying NFS load on the monitored system. You see a `NFS Client V2 & Server V2 Request Traffic for host` *host* window similar to the one shown in Figure 26.

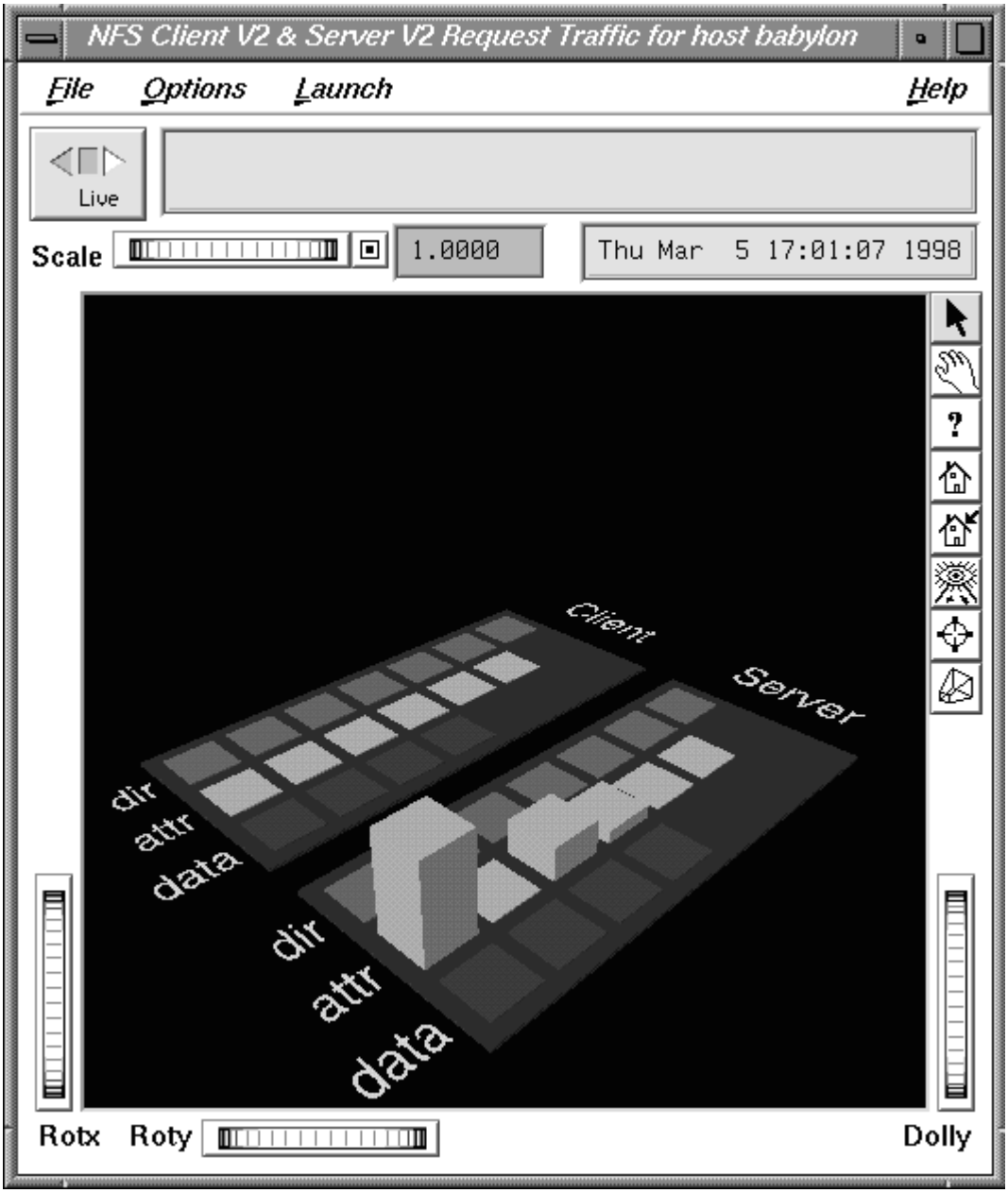

Figure 26. `nfsvis` `NFS Client V2 & Server V2 Request Traffic` Window

The statistics are broken into two groups:

`Client`	Requests made by the monitored machine to NFS servers on other machines
`Server`	Requests from other machines for the NFS server on the machine being monitored

The statistics in each of these two groups are the same, except that the client group is for outgoing requests and the server group is for incoming requests. Within each group, the requests are further broken down into three categories:

- Requests relating to data within files
- Requests for directory operations (for example, to rename a file)
- Requests involving other attributes of files

The `nfsvis` implementation uses the generalized 3D performance viewer `pmview` as described in Section 5.7. Hence, the command line options for `nfsvis` include the common ones for `pmview`.

Complete information on the `nfsvis` command is available in the `nfsvis`(1) man page. The PCP Tutorial contains additional examples on the use of `nfsvis`.

5.7 The `pmview` Tool

The `pmview` tool is a generalized three-dimension Open Inventor application that supports dynamic displays of clusters of related performance metrics as utilization blocks (or towers) on a common base plane. The `pmview` tool is the basis for `dkvis`, `mpvis`, `osvis`, and `nfsvis`, all discussed above, as well as `nodevis`, `routervis`, `txmonvis`, and `xbowvis`. It is also used by a range of related tools that are specific to PCP add-on products. The `pmview` tool may also be used to construct customized 3D performance displays.

Open Inventor is an object-oriented toolkit that simplifies and abstracts the task of writing graphics applications into a set of easy-to-use objects. Inventor run-time support is distributed with IRIX system software in the `inventor_eoe.sw` product image.

The `pmview` command displays performance metrics as colored blocks arranged in a grid on a grey base plane. The height of each block changes as the value of its corresponding metric (or metric instance) changes. Labels may be added to the scene to help identify groups of metrics, as shown in Figure 22, Figure 23, Figure 24, and Figure 26.

A configuration file is used to specify the position, color, and scale of metrics and metric instances in the scene. Metric values that exceed the associated scaling factor are displayed at the maximum height and change color to white. If a metric is unavailable, the bar height is minimized, and the bar color changes to grey.

Normally, `pmview` operates in live mode where performance metrics are fetched in real time. The user can view metrics from other accessible Internet hosts that are running the PCP collector daemon, `pmcd`. The `pmview` tool can also replay archives of performance metrics collected by `pmlogger`.

All metrics in the PMNS with numeric value semantics from multiple hosts or archives may be visualized. The `pmview` tool examines the semantics of the metrics and, where sensible, converts the fetched metric values to a rate before scaling.

The `pmview` tool window contains a menu bar, time and scale controls, metric and time values, and an *examiner* window; see the `ivview` command, which displays the 3D scene.

The left, right, and bottom edges of the examiner viewer window contain a variety of thumb wheels and buttons that allow the user to adjust the visualization of the 3D scene. The `Rotx` and `Roty` thumb wheels allow the user to rotate the scene about the X and Y axes, respectively. The `Dolly` thumb wheel moves the virtual camera closer to or further from the scene, allowing the user to examine specific parts in detail or view the entire scene.

On the right edge of the viewer are eight buttons that affect the way the user can interact with the scene:

- The `pointer` button changes the cursor to a pointer that allows blocks in the scene to be selected. The `Esc` key can also be used to toggle between the pointer and hand cursors.
- The `hand` button changes the cursor to a hand that can be used to rotate, translate, and examine the scene via `Dolly`, using a combination of mouse buttons and movement.

  The left mouse button can be used to rotate the scene in the direction of the mouse. Releasing the mouse button before the mouse has stopped moving causes the scene to continue rotating until a mouse button is pressed again.

  The middle mouse button can be used to pan the scene. By pressing both left and middle buttons, the mouse can be used as a virtual camera.

- The `question mark` button displays `SGI Help` for the examiner viewer. To install online help, use `inst` to install the `inventor_eoe.sw.help` package from your IRIX system software distribution. See the Performance Co-Pilot release notes for more information on prerequisite subsystems.
- The `home` button changes the scene back to its original position, or the position set by the `home pointer` button.
- The `home pointer` button sets the new home position of the scene to be the scene currently in view.
- The `eye` button resizes the scene so that it completely fits into the 3D viewing area.
- The `cross-hairs` button moves the scene so that the object under the cursor is in the center of the viewing area. Change the hand cursor and press the `cross-hairs` button. The cursor changes to a target. Select the block to be centered and the scene rotates and translates appropriately.
- The `perspective box` button switches between perspective and orthogonal projections.

Pressing the right mouse button within the scene displays a menu of options that affect how the 3D scene is drawn. The options include drawing the blocks as wireframes and turning on stereo viewing.

When the pointer cursor is active, more information about the 3D scene can be obtained. Text describing the metric represented by the block beneath the cursor displays in the top text box of the `pmview` window. This text displays the source, name, and instance of the performance metric, and the value, units, and percentage of the expected minimum the value represents.

Clicking the left mouse button on a block highlights the block with a red wireframe, as shown in Figure 27. The metric description text box is now fixed on that metric and the values continue to be updated as new metrics are fetched. This allows other actions to be performed with the mouse while examining a single metric in detail at the same time. Click the left mouse button on the space surrounding the scene to remove the selection.

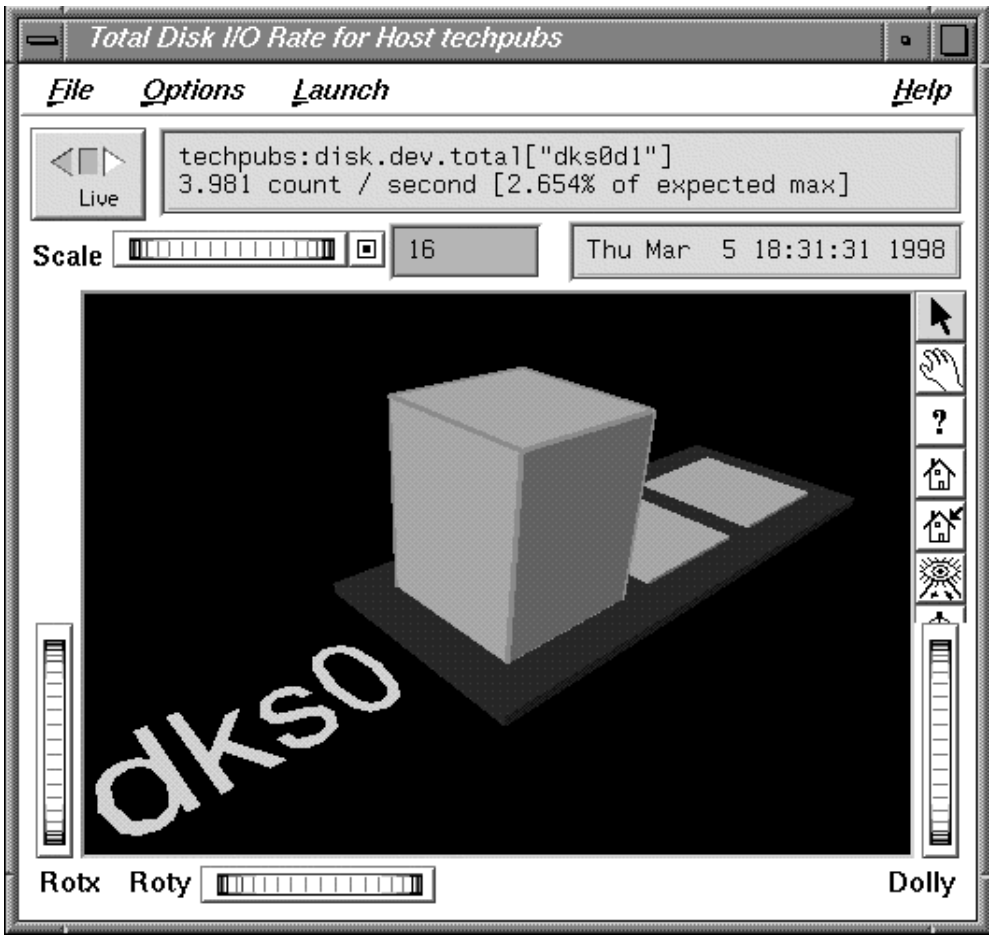

Figure 27. `pmview` Window with a Block Selected

Multiple blocks may also be selected by either Shift clicking with the left mouse button or by clicking on a base plane. Shift clicking toggles the selection status of a particular block and leaves the selection status of other blocks unaltered. Clicking on the base plane selects all blocks belonging to the base plane. Whenever multiple blocks are selected, no accompanying text is displayed in the text box. However, multiple block selection affects the launching of other tools because all metrics on the base plane are considered to be selected as a group.

5.7.1 pmview Menus

There are four menus in pmview tools:

File	Records, saves, and prints scenes
Options	Accesses the time controls
Launch	Starts other tools
Help	Obtains online help

The Launch menu consists of a list of tools that operate on the current selection of metrics. How the tool is invoked depends on the type of tool. Tools that operate on any metric (such as pmchart, pmval, and pmdumptext) use the metrics selected directly as input. Thus pmchart displays all the selected metrics in a chart, pmval is invoked within winterm for each metric, and pmdumptext displays multiple metrics in one winterm. Other tools use what metrics are pertinent. If no metric is pertinent or selected, only the source of the metrics is used, that is, the monitored host or archive. For each Launch menu item there is an associated launch script. The launch scripts generally know the relationship between routers, nodes, and CPUs. Thus if CPUs are selected in mpvis and if nodevis is launched, only the nodes that have the selected CPUs attached are displayed.

Some launchable tools are listed below:

dkvis, mpvis, nfsvis, and osvis	pmview-based tools for visualizing disk activity, CPUs, NFS, and the OS (operating system).
pcp	Brings up a window that summarizes the PCP installation.
pmdumptext	Brings up a window that shows the performance metrics as text.
pmchart	A tool for graphically displaying and correlating time-series trends of performance metrics. See Section 4.1, page 62, for details.
pmgsys	A miniature IRIX performance metrics viewer, available only in live mode, not in archive mode.
pmkstat	A text-based tool that displays, at intervals, a high-level summary of system performance.
pmval	A tool that displays the values of performance metrics textually. Only one metric (with one or more instances) may be selected to successfully

launch this tool. See Section 4.5, page 86, for details.

In addition to the menu options for time controls, the current direction and mode of the time controls is shown in a button in the top-left corner of the `pmview` window (refer to Section 3.4 for a complete description of the time control services). Pressing this button displays the time control dialog.

Below this button is a thumb wheel and an editable text box to specify a scale multiplier that is applied to all values in the scene. Spinning the thumb wheel to the right, or incrementing the value in the text field, increases the scaling and raises the height of the bars. Conversely, spinning the thumb wheel to the left or decrementing the text field decreases the scaling and lowers the height of the bars.

The button beside the thumb wheel resets the scale to one. This is especially useful when the scale specified in the configuration file reduces the usefulness of the visualization as a consequence of the bars being either too low or beyond the maximum scale height.

5.7.2 Creating Custom Visualization Tools with `pmview`

At startup time, a configuration file is read that specifies the following:

- Geometry for the scene to be displayed by `pmview`
- Associations between the visual appearance of "blocks" and performance metrics

The scene is based on a grid that can contain a variety of objects and can resize itself to accommodate objects of varying sizes. To distinguish this configuration file format from an earlier (still supported) format, configuration files must begin with the following line:

```
pmview Version 2.1
```

All lines beginning with a # character are treated as comments and ignored. Spaces, tabs, and newlines are treated as white space to allow multiple statements on the same line. The simplest configuration file consists of a single object that may represent one or more metrics and metric instances.

The configuration file consists of two sections: global parameters and color lists, and the object definitions. The global parameters control the size of the objects in the scene. For example, a scaling factor of 1.2 can be applied to all objects with the following line:

```
_scale 1.2
```

Groups of colors may be associated with a name and referenced later in the file. Colors may be `X(1` color names, `X(1)` numerical colors, or three real values representing the saturation of red, green, and blue, respectively. The following color list contains three identical colors:

```
_colorlist cpu ( red rgbi:1.0/0.0/0.0 1.0 0.0 0.0 )
```

The `mpvis` configuration file (which can be generated with the `-V` option) looks like Example 3, page 113:

Example 3: `mpvis` Configuration File

```
pmview Version 2.1
#
# mpvis
#
_gridSpace 120
_colorlist cpu ( green2 cyan2 yellow2 red2 blue2 )
_grid 0 0 _hide ( # outer grid
            _baseLabel ‘‘CPU Utilization for Host wired\ncpu0 only’’
            _bar _groupByInst (
               _metrics (
                   kernel.percpu.cpu.idle[cpu0]        1000 ‘‘idle’’
                   kernel.percpu.cpu.wait.total[cpu0]  1000 ‘‘wait’’
                   kernel.percpu.cpu.intr[cpu0]        1000 ‘‘intr’’
                   kernel.percpu.cpu.sys[cpu0]         1000 ‘‘sys’’
                   kernel.percpu.cpu.user[cpu0]        1000 ‘‘user’’
               )
               _colorlist cpu
               _baseLabel ‘‘CPU Utilization for Host wired\ncpu0 only’’
           )
)
```

Multiple objects can be visualized using a `_grid` object, which may contain multiple objects (including more `_grid` objects). The `_grid` object resizes columns and rows to accommodate the largest contained object. Objects can occupy multiple grid squares and can be aligned with a particular edge or corner of a grid square. The `_bar` object has a single bar for each metric instance and labels for each metric. The scale height for each metric instance is 1000 in the units of the metric (milliseconds utilization per second).

The specification file shown in Example 4, page 114, produces a scene like the one shown in Figure 28. The file has a grid, labels, bars, and a stack utilization object.

Example 4: Specification File for `pmview`

```
pmview Version 2.1
_colorlist cpu_colors ( blue2 red2 yellow2 cyan2 green2 )
_colorlist disk_colors ( purple2 yellow2 )
_colorlist memory_colors ( rgbi:1.0/1.0/0.0 rgbi:0.0/1.0/1.0 rgbi:1.0/0.0/0.0
                           rgbi:1.0/0.0/1.0 rgbi:0.0/0.0/1.0 rgbi:0.0/1.0/0.0 )
_grid hide (
_label 3 1 _west _down _large ‘‘CPU’’
       _stack 4 1 _west _utilmod (
             _metrics (
               kernel.all.cpu.user        1000
               kernel.all.cpu.sys         1000
               kernel.all.cpu.intr        1000
               kernel.all.cpu.wait.total  1000
               kernel.all.cpu.idle        1000
           )
           _colorlist cpu_colors
           _baseLabel ‘‘CPU Utilization’’
       )
_label 3 3 _west _down _large ‘‘Load’’
       _bar 4 3 2 1 _west (
           _metrics (
               kernel.all.load[15]    2
               kernel.all.load[5]     2
               kernel.all.load[1]     2
           )
           _metriclabels _away ( ‘‘15’’ ‘‘5’’ ‘‘1’’ )
           _colorlist ( blue2 blue2 blue2 )
_baseLabel ‘‘Average System Load over last 1, 5 and 15 minutes\nNormalized to 2’’
       )
_label 0 1 _west _down _large ‘‘Mem’’
         _stack 1 1 _west _utilmod (
           _metrics (
               mem.util.kernel        1
               mem.util.fs_ctl        1
               mem.util.fs_dirty      1
               mem.util.fs_clean      1
               mem.util.user          1
```

```
            )
            _colorlist memory_colors
            _baseLabel ‘‘Physical Memory Utilization’’
        )
        _label 0 3 _down _large ‘‘Disk’’
        _stack 1 3 _west _cylinder (
            _metrics (
                disk.all.read          100
                disk.all.write         100
            )
            _colorlist disk_colors
            _baseLabel ‘‘Disk Operations\nNormalized to 100 I/Os per second’’
         )
)
```

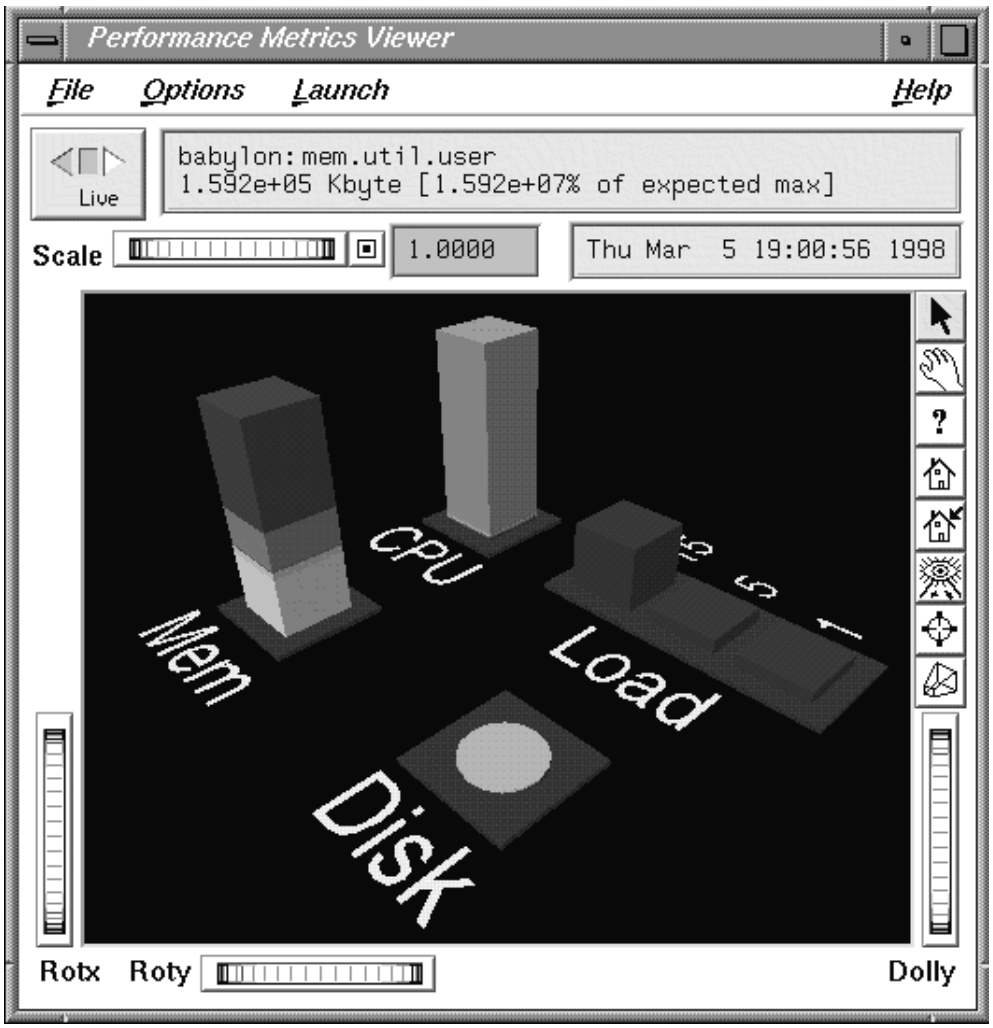

Figure 28. Custom `pmview` Scene

To assist in the creation of front-end tools, a file containing shell procedures for generating usage information and parsing `pmview` command line options is located at `/var/pcp/lib/pmview-args`. The `dkvis`, `mpvis`, `nfsvis`, and `osvis` tools are all shell scripts that use these shell procedures to generate a configuration file for `pmview`.

Performance Metrics Inference Engine [6]

The Performance Metrics Inference Engine (`pmie`) is a tool that provides automated monitoring of, and reasoning about, system performance within the Performance Co-Pilot (PCP) framework.

The following major sections in this chapter are as follows:

- Section 6.1, page 117, provides an introduction to the concepts and design of `pmie`.
- Section 6.2, page 120, describes the basic syntax and usage of `pmie`.
- Section 6.3, page 124, discusses the complete `pmie` rule specification language.
- Section 6.4, page 139, provides an example, covering several common performance scenarios.
- Section 6.5, page 141, presents some tips and techniques for `pmie` rule development.
- Section 6.6, page 141, presents some important information on using `pmie`.
- Section 6.7, page 143, describes how to use the `pmieconf` command to generate `pmie` rules.
- Section 6.8, page 146, provides a brief description of how to use the `pmrules` GUI for creating `pmie` rules from parameterized templates.
- Section 6.9, page 150, provides support for running `pmie` as a daemon.

6.1 Introduction to `pmie`

Automated reasoning within Performance Co-Pilot (PCP) is provided by the Performance Metrics Inference Engine, (`pmie`), which is an applied artificial intelligence application.

The `pmie` tool accepts expressions describing adverse performance scenarios, and periodically evaluates these against streams of performance metric values from one or more sources. When an expression is found to be true, `pmie` is able to execute arbitrary actions to alert or notify the system administrator of the occurrence of an adverse performance scenario. These facilities are very general,

and are designed to accommodate the automated execution of a mixture of generic and site-specific performance monitoring and control functions.

The stream of performance metrics to be evaluated may be from one or more hosts, or from one or more PCP archive logs. In the latter case, `pmie` may be used to retrospectively identify adverse performance conditions.

Using `pmie`, you can filter, interpret, and reason about the large volume of performance data made available by the Performance Metrics Collection System (PMCS) and delivered through the Performance Metrics Application Programming Interface (PMAPI).

Typical `pmie` uses include the following:

- Automated real-time monitoring of a host, a set of hosts, or client-server pairs of hosts to raise operational alarms when poor performance is detected in a production environment
- Nightly processing of archive logs to detect and report performance regressions, or quantify quality of service for service agreements or management reports, or produce advance warning of pending performance problems
- Strategic performance management, for example, detection of abnormal, but not chronic, system behavior, trend analysis, and capacity planning

The `pmie` expressions are described in a language with expressive power and operational flexibility. It includes the following operators and functions:

- Generalized predicate-action pairs, where a predicate is a logical expression over the available performance metrics, and the action is arbitrary. Predefined actions include the following:
  - Launch a visible alarm with `xconfirm`; see the `xconfirm`(1) man page.
  - Post an entry to the system log `/var/adm/SYSLOG`; see the `syslog`(3C) man page.
  - Post an entry to the PCP noticeboard file `/var/adm/pcplog/NOTICES`.
  - Execute a shell command or script, for example, to send e-mail, initiate a pager call, warn the help desk, and so on.
  - Echo a message on standard output; useful for scripts that generate reports from retrospective processing of PCP archive logs.
- Arithmetic and logical expressions in a C-like syntax.

- Expression groups may have an independent evaluation frequency, to support both short-term and long-term monitoring.
- Canonical scale and rate conversion of performance metric values to provide sensible expression evaluation.
- Aggregation functions of `sum`, `avg`, `min`, and `max`, that may be applied to collections of performance metrics values clustered over multiple hosts, or multiple instances, or multiple consecutive samples in time.
- Universal and existential quantification, to handle expressions of the form "for every...." and "at least one...".
- Percentile aggregation to handle statistical outliers, such as "for at least 80% of the last 20 samples, ...".
- Macro processing to expedite repeated use of common subexpressions or specification components.
- Transparent operation against either live-feeds of performance metric values from `pmcd` on one or more hosts, or against PCP archive logs of previously accumulated performance metric values.

The power of `pmie` may be harnessed to automate the most common of the deterministic system management functions that are responses to changes in system performance. For example, disable a batch stream if the DBMS transaction commit response time at the ninetieth percentile goes over two seconds, or stop accepting news and send e-mail to the *sysadmin* alias if free space in the news file system falls below five percent.

Moreover, the power of `pmie` can be directed towards the exceptional and sporadic performance problems. For example, if a network packet storm is expected, enable IP header tracing for ten seconds, and send e-mail to advise that data has been collected and is awaiting analysis. Or, if production batch throughput falls below 50 jobs per hour, activate a pager to the systems administrator on duty.

Obviously, `pmie` customization is required to produce meaningful filtering and actions in each production environment. The `pmieconf` tool provides a convenient customization method, allowing the user to generate parameterized `pmie` rules for some of the more common performance scenarios.

6.2 Basic pmie Usage

This section presents and explains some basic examples of pmie usage. The pmie tool accepts the common PCP command line arguments, as described in Chapter 3, page 43. In addition, pmie accepts the following command line arguments:

-d	Enables interactive debug mode.
-v	Verbose mode: expression values are displayed.
-V	Verbose mode: annotated expression values are displayed.
-W	When-verbose mode: when a condition is true, the satisfying expression bindings are displayed.

One of the most basic invocations of this tool is this form:

pmie *filename*

In this form, the expressions to be evaluated are read from filename. In the absence of a given filename, expressions are read from standard input, usually your system keyboard.

6.2.1 pmie and the Performance Metrics Collection System

Before you use pmie, familiarize yourself with some Performance Metrics Collection System (PMCS) basics. It is strongly recommended that you familiarize yourself with the concepts from the Section 1.3, page 13. The discussion in this section serves as a very brief review of these concepts.

The PMCS makes available hundreds of performance metrics that you can use when formulating expressions for pmie to evaluate. If you want to find out which metrics are currently available on your system, use this command:

```
pminfo
```

Use the pmie command line arguments to find out more about a particular metric. For example, to fetch new metric values from host moomba, use the -f flag:

```
pminfo -f -h moomba disk.dev.total
```

This produces the following response:

```
disk.dev.total
    inst [131329 or "dks1d1"] value 970853
    inst [131330 or "dks1d2"] value 53581
```

```
    inst [131331 or "dks1d3"] value 5353
    inst [131332 or "dks1d4"] value 225
    inst [131333 or "dks1d5"] value 9674
    inst [131334 or "dks1d6"] value 14383
    inst [131335 or "dks1d7"] value 5578
```

This reveals that on the host `moomba`, the metric `disk.dev.total` has seven instances, one for each disk on the system. The instance names are `dks1d1`, `dks1d2`, and so on up to `dks1d7`.

Use the following command to request help text (specified with the `-T` flag) to provide more information about performance metrics:

`pminfo -T network.interface.in.packets`

The metadata associated with a performance metric is used by `pmie` to determine how the value should be interpreted. You can examine the descriptor that encodes the metadata by using the `-d` flag for `pminfo`, as shown in this command:

`pminfo -d -h` *somehost* **`mem.freemem kernel.percpu.syscall`**

In response, you see output similar to this:

```
mem.freemem
       Data Type: 32-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
       Semantics: instant  Units: Kbyte
kernel.percpu.syscall
       Data Type: 32-bit unsigned int  InDom: 1.1 0x400001
       Semantics: counter  Units: count
```

Note: A cumulative counter such as `kernel.percpu.syscall` is automatically converted by `pmie` into a rate (measured in events per second, or count/second), while instantaneous values such as `mem.freemem` are not subjected to rate conversion. Metrics with an instance domain (InDom in the `pminfo` output) of `PM_INDOM_NULL` are singular and always produce one value per source. However, a metric like `kernel.percpu.syscall` has an instance domain, and may produce multiple values per source (in this case, it is one value for each configured CPU).

6.2.2 Simple `pmie` Example

The following `pmie` example directs the inference engine to evaluate and print values (specified with the `-v` flag) for a single performance metric (the simplest

possible expression), in this case `disk.dev.total`, collected from the local `pmcd`:

```
pmie -v
iops = disk.dev.total;
Ctrl+D
iops:      ?      ?
iops:   14.4      0
iops:   25.9  0.112
iops:   12.2      0
iops:   12.3   64.1
iops:  8.594  52.17
iops:  2.001  71.64
```

On this system, there are two disk spindles, hence two values of the expression `iops` per sample. Notice that the values for the first sample are unknown (represented by the question marks [?] in the first line of output), because rates can be computed only when at least two samples are available. The subsequent samples are produced every ten seconds by default. The second sample reports that during the preceding ten seconds there was an average of 14.4 transfers per second on one disk and no transfers on the other disk.

Rates are computed using time stamps delivered by the PMCS. Due to unavoidable inaccuracy in the actual sampling time (the sample interval is not exactly 10 seconds), you may see more decimal places in values than you expect. Notice, however, that these errors do not accumulate but cancel each other out over subsequent samples.

In the above example, the expression to be evaluated was enter (the keyboard), followed by the end-of-file character [Ctrl+D]. Usually, it is more convenient to enter expressions into a file (for example, `myrules`) and ask `pmie` to read the file. Use this command syntax:

```
pmie -v myrules
```

Please refer to the `pmie`(1) man page for a complete description of `pmie` command line options.

6.2.3 Complex `pmie` Examples

This section illustrates more complex `pmie` expressions of the specification language. The next section provides a complete description of the `pmie` specification language.

The following arithmetic expression computes the percentage of write operations over the total number of disk transfers.

```
(disk.all.write / disk.all.total) * 100;
```

The `disk.all` metrics are singular, so this expression produces exactly one value per sample, independent of the number of disk devices.

Note: If there is no disk activity, `disk.all.total` will be zero and `pmie` evaluates this expression to be not a number. When `-v` is used, any such values are displayed as question marks.

The following logical expression has the value `true` or `false` for each disk:

```
disk.dev.total > 10 &&
disk.dev.write > disk.dev.read;
```

The value is true if the number of writes exceeds the number of reads, and if there is significant disk activity (more than 10 transfers per second).

The previous examples did not specify any action to be performed in the event that an expression evaluates to `true`. The default action is to do nothing, other than report the value of the expression if the `-v` option was used. The following example demonstrates a simple action:

```
some_inst disk.dev.total > 60 ->
                                print "[%i] high disk i/o ";
```

This prints a message to the standard output whenever the total number of transfers for some disk (`some_inst`) exceeds 60 transfers per second. The `%i` (instance) in the message is replaced with the name(s) of the disk(s) that caused the logical expression to be `true`.

Using `pmie` to evaluate the above expressions every 3 seconds, you see output similar to the following:

```
pmie -v -t 3sec
pct_wrt = (disk.all.write / disk.all.total) * 100;
busy_wrt = disk.dev.total > 10 &&
           disk.dev.write > disk.dev.read;
busy = some_inst disk.dev.total > 60 ->
                          print "[%i] high disk i/o ";
Ctrl+D
pct_wrt:       ?
busy_wrt:      ?      ?
busy:          ?
```

```
pct_wrt:   18.43
busy_wrt:  false  false
busy:      false

Mon Aug  5 14:56:08 1996: [dks0d2] high disk i/o
pct_wrt:   10.83
busy_wrt:  false  false
busy:      true

pct_wrt:   19.85
busy_wrt:   true  false
busy:      false

pct_wrt:       ?
busy_wrt:  false  false
busy:      false

Mon Aug  5 14:56:17 1996: [dks0d1] high disk i/o [dks0d2] high disk i/o
pct_wrt:   14.8
busy_wrt:  false  false
busy:   true
```

The first sample contains unknowns, since all expressions depend on computing rates. Also notice that the expression `pct_wrt` may have an undefined value whenever all disks are idle, as the denominator of the expression is zero. If one or more disks is busy, the expression `busy` is true, and the message from the `print` in the action part of the rule appears (before the `-v` values).

6.3 Specification Language for `pmie`

This section describes the complete syntax of the `pmie` specification language, as well as macro facilities and the issue of sampling and evaluation frequency. The reader with a preference for learning by example may choose to skip this section and go straight to the examples in Section 6.4, page 139.

Complex expressions are built up recursively from simple elements:

1. Performance metric values are obtained from `pmcd` for real-time sources, otherwise from PCP archive logs.
2. Metrics values may be combined using arithmetic operators to produce arithmetic expressions.

3. Arithmetic expressions may be compared using relational operators to produce logical expressions.
4. Logical expressions may be combined using Boolean operators, including powerful quantifiers.
5. Aggregation operators may be used to compute summary expressions, for either arithmetic or logical operands.
6. The final logical expression may be used to initiate a sequence of actions.

6.3.1 Basic `pmie` Syntax

The `pmie` rule specification language supports a number of basic syntactic elements.

6.3.1.1 Lexical Elements

All `pmie` expressions are composed of the following lexical elements:

Identifier	Begins with an alphabetic character (either upper or lowercase), followed by zero or more letters, the numeric digits, and the special characters period (`.`) and underscore (_), as shown in the following example:
	`x, disk.dev.total and my_stuff`
	As a special case, an arbitrary sequence of letters enclosed by apostrophes (`'`) is also interpreted as an *identifier*; for example:
	`'vms$slow_response'`

Keyword	The aggregate operators, units, and predefined actions are represented by keywords; for example, `some_inst`, `print`, and `hour`.
Numeric constant	Any likely representation of a decimal integer or floating point number; for example, 124, 0.05, and -45.67

String constant	An arbitrary sequence of characters, enclosed by double quotation marks (`"x"`).

Within quotes of any sort, the backslash (`/`) may be used as an escape character as shown in the following example:

```
"A \"gentle\" reminder"
```

6.3.1.2 Comments

Comments may be embedded anywhere in the source, in either of these forms:

`/* text */`	Comment, optionally spanning multiple lines, with no nesting of comments.
`// text`	Comment from here to the end of the line.

6.3.1.3 Macros

When they are fully specified, expressions in `pmie` tend to be verbose and repetitious. The use of macros can reduce repetition and improve readability and modularity. Any statement of the following form associates the macro name `identifier` with the given `string` constant.

identifier = "string";

Any subsequent occurrence of the macro name `identifier` is replaced by the *string* most recently associated with a macro definition for *identifier*.

$*identifier*

For example, start with the following macro definition:

```
disk = "disk.all";
```

You can then use the following syntax:

```
pct_wrt = ($disk.write / $disk.total) * 100;
```

Note: Macro expansion is performed before syntactic parsing; so macros may only be assigned constant string values.

6.3.1.4 Units

The inference engine converts all numeric values to canonical units (*seconds* for time, *bytes* for space, and *events* for count). To avoid surprises, you are encouraged to specify the units for numeric constants. If units are specified,

they are checked for dimension compatibility against the metadata for the associated performance metrics.

The syntax for a `units` specification is a sequence of one or more of the following keywords separated by either a space or a slash (`/`), to denote per: `byte`, `KByte`, `MByte`, `GByte`, `TByte`, `nsec`, `nanosecond`, `usec`, `microsecond`, `msec`, `millisecond`, `sec`, `second`, `min`, `minute`, `hour`, `count`, `Kcount`, `Mcount`, `Gcount`, or `Tcount`. Plural forms are also accepted.

The following are examples of units usage:

```
disk.dev.blktotal > 1 Mbyte / second;
mem.freemem < 500 Kbyte;
```

Note: If you do not specify the units for numeric constants, it is assumed that the constant is in the canonical units of *seconds* for time, *bytes* for space, and *events* for count, and the dimensionality of the constant is assumed to be correct. Thus, in the following expression, the `500` is interpreted as 500 bytes.

```
mem.freemem < 500
```

6.3.2 Setting Evaluation Frequency

The identifier name `delta` is reserved to denote the interval of time between consecutive evaluations of one or more expressions. Set `delta` as follows:

`delta =` *number* `[`*units*`];`

If present, `units` must be one of the time units described in the preceding section. If absent, `units` are assumed to be `seconds`. For example,

```
delta = 5 min;
```

has the effect that any subsequent expressions (up to the next expression that assigns a value to `delta`) are scheduled for evaluation at a fixed frequency, once every five minutes.

The default value for *delta* may be specified using the `-t` command line option, otherwise *delta* is initially set to be 10 seconds.

6.3.3 `pmie` Metric Expressions

A Performance Metrics Name Space (PMNS) provides a means of naming performance metrics, for example, `disk.dev.read`. The Performance Metrics Collection System (PMCS) allows an application to retrieve one or more values

for a performance metric from a designated source (a collector host running `pmcd`, or a PCP archive log). To specify a single value for some performance metric requires the metric name to be associated with all three of the following:

- A particular host (or source of metrics values)
- A particular instance (for metrics with multiple values)
- A sample time

The permissible values for hosts are the range of valid hostnames as provided by Internet naming conventions.

The names for instances are provided by the Performance Metrics Domain Agents (PMDA) for the instance domain associated with the chosen performance metric.

The sample time specification is defined as the set of natural numbers 0, 1, 2, and so on. A number refers to one of a sequence of sampling events, from the current sample 0 to its predecessor 1, whose predecessor was 2, and so on. This scheme is illustrated by the time line shown in Figure 29.

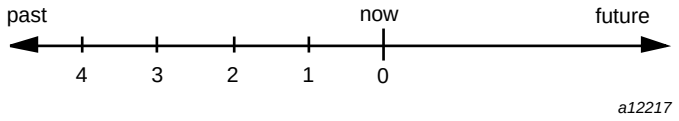

Figure 29. Sampling Time Line

Each sample point is assumed to be separated from its predecessor by a constant amount of real time, the *delta*. The most recent sample point is always zero. The value of *delta* may vary from one expression to the next, but is fixed for each expression; for more information on the sampling interval, see Section 6.3.2, page 127.

For `pmie`, a metrics expression is the name of a metric, optionally qualified by a host, instance and sample time specification. Special characters introduce the qualifiers: colon (`:`) for hosts, hash or pound sign (#) for instances, and at (`@`) for sample times. The following expression refers to the previous value (`@1`) of the counter for the disk read operations associated with the disk instance `#dks0d1` on the host `moomba`.

```
disk.dev.read :moomba #dks0d1 @1
```

In fact, this expression defines a point in the three-dimensional parameter space of {`host`} x {`instance`} x {`sample time`} as shown in Figure 30.

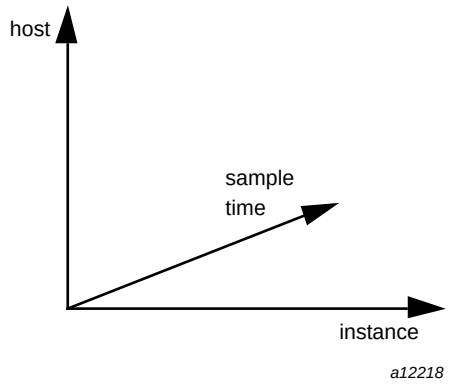

Figure 30. Three-Dimensional Parameter Space

A metric expression may also identify sets of values corresponding to one-, two-, or three-dimension slices of this space, according to the following rules:

1. A metric expression consists of a PCP metric name, followed by optional *host* specifications, followed by optional *instance* specifications, and finally, optional *sample time* specifications.
2. A *host* specification consists of one or more host names, each prefixed by a colon (`:`). For example: `:indy :far.away.domain.com :localhost`
3. A missing *host* specification implies the default `pmie` source of metrics, as defined by a `-h` option on the command line, or the first named archive in a `-a` option on the command line, or `pmcd` on the local host.
4. An *instance* specification consists of one or more instance names, each prefixed by a hash or pound (#) sign. For example: `#ec0 #ec2`

   Recall that you can discover the instance names for a particular metric, using the `pminfo` command. See Section 6.2.1, page 120.

   Within the `pmie` grammar, an instance name is an identifier. If the instance name contains characters other than alphanumeric characters, enclose the instance name in single quotes; for example, `#'/dev/root' #'/dev/usr'`

5. A missing *instance* specification implies all instances for the associated performance metric from each associated `pmie` source of metrics.
6. A *sample time* specification consists of either a single time or a range of times. A single time is represented as an at (`@`) followed by a natural number. A range of times is an at (`@`), followed by a natural number, followed by two periods (`..`) followed by a second natural number. The ordering of the end points in a range is immaterial. For example, `@0..9` specifies the last 10 sample times.
7. A missing *sample time* specification implies the most recent sample time.

The following metric expression refers to a three-dimension set of values, with two hosts in one dimension, five sample times in another, and the number of instances in the third dimension being determined by the number of configured disk spindles on the two hosts.

```
disk.dev.read :foo :bar @0..4
```

6.3.4 `pmie` Rate Conversion

Many of the metrics delivered by the PMCS are cumulative counters. Consider the following metric:

```
disk.all.total
```

A single value for this metric tells you only that a certain number of disk I/O operations have occurred since boot time, and that information may be invalid if the counter has exceeded its 32-bit range and wrapped. You need at least two values, sampled at known times, to compute the recent rate at which the I/O operations are being executed. The required syntax would be this:

```
(disk.all.total @0 - disk.all.total @1) / delta
```

The accuracy of *delta* as a measure of actual inter-sample delay is an issue. `pmie` requests samples, at intervals of approximately *delta*, while the results exported to the PMCS are time stamped with the high-resolution system clock time when the samples were exported. For these reasons, a built-in and implicit rate conversion using accurate time stamps is provided by `pmie` for performance metrics that have counter semantics. For example, the following expression is unconditionally converted to a rate by `pmie`.

```
disk.all.total
```

6.3.5 pmie Arithmetic Expressions

Within pmie, simple arithmetic expressions are constructed from metrics expressions (see Section 6.3.3, page 127) and numeric constants, using all of the arithmetic operators and precedence rules of the C programming language.

All pmie arithmetic is performed in double precision.

Section 6.3.8, page 137, describes additional operators that may be used for aggregate operations to reduce the dimensionality of an arithmetic expression.

6.3.6 pmie Logical Expressions

A number of logical expression types are supported:

- Logical constants
- Relational expressions
- Boolean expressions
- Quantification operators

6.3.6.1 Logical Constants

Like in the C programming language, pmie interprets an arithmetic value of zero to be false, and all other arithmetic values are considered true.

6.3.6.2 Relational Expressions

Relational expressions are the simplest form of logical expression, in which values may be derived from arithmetic expressions using pmie relational operators. For example, the following is a relational expression that is true or false, depending on the aggregate total of disk read operations per second being greater than 50.

```
disk.all.read > 50 count/sec
```

All of the relational logical operators and precedence rules of the C programming language are supported in pmie.

As described in Section 6.3.3, page 127, arithmetic expressions in pmie may assume set values. The relational operators are also required to take constant, singleton, and set-valued expressions as arguments. The result has the same dimensionality as the operands. Suppose the following rule is given:

```
hosts = ":gonzo";
intfs = "#ec0 #ec2";
all_intf = network.interface.in.packets
                $hosts $intfs @0..2 > 300 count/sec;
```

Then the execution of `pmie` may proceed as follows:

```
pmie -V uag.11
all_intf:
    gonzo: [ec0]      ?      ?      ?
    gonzo: [ec2]      ?      ?      ?
all_intf:
    gonzo: [ec0]  false      ?      ?
    gonzo: [ec2]  false      ?      ?
all_intf:
    gonzo: [ec0]   true  false      ?
    gonzo: [ec2]  false  false      ?
all_intf:
    gonzo: [ec0]   true   true  false
    gonzo: [ec2]  false  false  false
```

At each sample, the relational operator greater than (>) produces six truth values for the cross-product of the *instance* and *sample time* dimensions.

Section 6.3.6.4, page 132, describes additional logical operators that may be used to reduce the dimensionality of a relational expression.

6.3.6.3 Boolean Expressions

The regular Boolean operators from the C programming language are supported: conjunction (&&), disjunction (||) and negation (!).

As with the relational operators, the Boolean operators accommodate set-valued operands, and set-valued results.

6.3.6.4 Quantification Operators

Boolean and relational operators may accept set-valued operands and produce set-valued results. In many cases, rules that are appropriate for performance management require a set of truth values to be reduced along one or more of the dimensions of hosts, instances, and sample times described in Section 6.3.3, page 127. The `pmie` quantification operators perform this function.

Each quantification operator takes a one-, two-, or three-dimension set of truth values as an operand, and reduces it to a set of smaller dimension, by

quantification along a single dimension. For example, suppose the expression in the previous example is simplified and prefixed by `some_sample`, to produce the following expression:

```
intfs = "#ec0 #ec2";
all_intf = some_sample network.interface.in.packets
                    $intfs @0..2 > 300 count/sec;
```

Then the expression result is reduced from six values to two (one per interface instance), such that the result for a particular instance will be false unless the relational expression for the same interface instance is true for at least one of the preceding three sample times.

There are existential, universal, and percentile quantification operators in each of the *host*, *instance,* and *sample time* dimensions to produce the nine operators as follows:

`some_host`	True if the expression is true for at least one *host* for the same *instance* and *sample time*.
`all_host`	True if the expression is true for every *host* for the same *instance* and *sample time*.
N`%_host`	True if the expression is true for at least *N*% of the *hosts* for the same *instance* and *sample time*.
`some_inst`	True if the expression is true for at least one *instance* for the same *host* and *sample time*.
`all_instance`	True if the expression is true for every *instance* for the same *host* and *sample time*.
N`%_instance`	True if the expression is true for at least *N*% of the *instances* for the same *host* and *sample time*.
`some_sample time`	True if the expression is true for at least one *sample time* for the same *host* and *instance*.
`all_sample time`	True if the expression is true for every *sample time* for the same *host* and *instance*.
N`%_sample time`	True if the expression is true for at least *N*% of the *sample times* for the same *host* and *instance*.

These operators may be nested. For example, the following expression answers the question: “Are all hosts experiencing at least 20% of their disks busy either reading or writing?”

```
Servers = ":moomba :babylon";
all_host (
    20%_inst disk.dev.read $Servers > 40 ||
    20%_inst disk.dev.write $Servers > 40
);
```

The following expression uses different syntax to encode the same semantics:

```
all_host (
    20%_inst (
        disk.dev.read $Servers > 40 ||
        disk.dev.write $Servers > 40
    )
);
```

Note: To avoid confusion over precedence and scope for the quantification operators, use explicit parentheses.

Two additional quantification operators are available for the instance dimension only, namely `match_inst` and `nomatch_inst`, that take a regular expression and a boolean expression. The result is the boolean AND of the expression and the result of matching (or not matching) the associated instance name against the regular expression.

For example, this rule evaluates error rates on various 10BaseT Ethernet network interfaces (such as ecN, etN, or efN):

```
some_inst
                  match_inst "^(ec|et|ef)"
                      network.interface.total.errors > 10 count/sec
-> syslog "Ethernet errors:" " %i"
```

6.3.7 `pmie` Rule Expressions

Rule expressions for `pmie` have the following syntax:

```
lexpr -> actions ;
```

The semantics are as follows:

- If the logical expression `lexpr` evaluates `true`, then perform the *actions* that follow. Otherwise, do not perform the *actions*.
- It is required that `lexpr` has a singular truth value. Aggregation and quantification operators must have been applied to reduce multiple truth values to a single value.

- When executed, an *action* completes with a success/failure status.
- One or more *actions* may appear; consecutive *actions* are separated by operators that control the execution of subsequent *actions*, as follows:

action-1&	Always execute subsequent actions (serial execution).
action-1 |	If *action-1* fails, execute subsequent actions, otherwise skip the subsequent actions (alternation).

An *action* is composed of a keyword to identify the action method, an optional *time* specification, and one or more *argument*s.

A *time* specification uses the same syntax as a valid time interval that may be assigned to *delta*, as described in Section 6.3.2, page 127. If the *action* is executed and the *time* specification is present, `pmie` will suppress any subsequent execution of this *action* until the wall clock time has advanced by *time*.

The *argument*s are passed directly to the action method.

The following action methods are provided:

`shell`	The single *argument* is passed to the shell for execution. This *action* is implemented using `system` in the background. The *action* does not wait for the system call to return, and succeeds unless the fork fails.
`alarm`	A notifier containing a time stamp, a single *argument* as a message, and a `Cancel` button is posted on the current display screen (as identified by the `DISPLAY` environment variable). Each alarm *action* first checks if its notifier is already active. If there is an identical active notifier, a duplicate notifier is not posted. The action succeeds unless the fork fails.
`syslog`	A message is written into the system log as a priority (see the `-p` option for `pmlogger`);" to: "A message is written into the system log. If the first word of the first argument is `-p`, the second word is interpreted as the priority (see the `syslog`(3) man page)"; the message tag is `pcp-pmie`. The remaining *argument* is the message to be written

to the system log. The action succeeds unless the fork fails.

`print` — A message containing a time stamp in `ctime` format and the *argument* is displayed out to standard output (`stdout`). This action always succeeds.

Within the *argument* passed to an action method, the following expansions are supported to allow some of the context from the logical expression on the left to appear to be embedded in the *argument*:

`%h` — The value of a *host* that makes the expression true.

`%i` — The value of an *instance* that makes the expression true.

`%v` — The value of a performance metric from the logical expression.

Some ambiguity may occur in respect to which *host*, *instance,* or performance metric is bound to a %-token. In most cases, the leftmost binding in the top-level subexpression is used. You may need to use `pmie` in the interactive debugging mode (specify the `-d` command line option) in conjunction with the `-W` command line option to discover which subexpressions contributes to the %-token bindings.

The following example illustrates some of the options when constructing rule expressions:

```
some_inst ( disk.dev.total > 60 )
       -> syslog 10 mins "[%i] busy, %v IOPS " &
          shell 1 hour "echo \
               'Disk %i is REALLY busy. Running at %v I/Os per second' \
               | Mail -s 'pmie alarm' sysadm";
```

In this case, `%v` and `%i` are both associated with the instances for the metric `disk.dev.total` that make the expression true. If more than one instance makes the expression true (more than one disk is busy), then the *argument* is formed by concatenating the result from each %-token binding. For example, the text added to `/var/adm/SYSLOG` might be as follows:

```
Aug 6 08:12:44 5B:gonzo pcp-pmie[3371]:
                        [dks0d1] busy, 3.7 IOPS [dks0d2] busy, 0.3 IOPS
```

Note: When `pmie` is processing performance metrics from a PCP archive log, the *actions* will be processed in the expected manner; however, the action methods are modified to report a textual facsimile of the *action* on the standard output. For example, consider the following rule:

```
delta = 2 sec;  // more often for demonstration purposes
percpu  = "kernel.percpu";
// Unusual usr-sys split when some CPU is more than 20% in usr mode
// and sys mode is at least 1.5 times usr mode
//
cpu_usr_sys = some_inst (
        $percpu.cpu.sys > $percpu.cpu.user * 1.5 &&
        $percpu.cpu.user > 0.2
   ) ->  alarm "Unusual sys time: " "%i ";
```

When evaluated against an archive, the following output is generated (the alarm action produces a message on standard output):

```
pmafm /tmp/f4 pmie cpu.head cpu.00
alarm Wed Aug  7 14:54:48 1996: Unusual sys time: cpu0
alarm Wed Aug  7 14:54:50 1996: Unusual sys time: cpu0
alarm Wed Aug  7 14:54:52 1996: Unusual sys time: cpu0
alarm Wed Aug  7 14:55:02 1996: Unusual sys time: cpu0
alarm Wed Aug  7 14:55:06 1996: Unusual sys time: cpu0
```

6.3.8 `pmie` Intrinsic Operators

The following sections describe some other useful intrinsic operators for `pmie`. These operators are divided into three groups:

- Arithmetic aggregation
- The `rate` operator
- Transitional operators

6.3.8.1 Arithmetic Aggregation

For set-valued arithmetic expressions, the following operators reduce the dimensionality of the result by arithmetic aggregation along one of the *host*, *instance*, or *sample time* dimensions. For example, to aggregate in the *host* dimension, the following operators are provided:

`avg_host`	Computes the average value across all *instances* for the same *host* and *sample time*
`sum_host`	Computes the total value across all *instances* for the same *host* and *sample time*

`count_host`	Computes the number of values across all *instances* for the same *host* and *sample time*
`min_host`	Comutes the minimum value across all *instances* for the same *host* and *sample time*
`max_host`	Computes the maximum value across all *instances* for the same *host* and *sample time*

Ten additional operators correspond to the forms `*_inst` and `*_sample`.

The following example illustrates the use of an aggregate operator in combination with an existential operator to answer the question "Does some host currently have two or more busy processors?"

```
// note '' to escape - in host name
poke = ":moomba :'mac-larry' :bitbucket";
some_host (
    count_inst ( kernel.percpu.cpu.user $poke +
                 kernel.percpu.cpu.sys $poke > 0.7 ) >= 2
    )
       -> alarm "2 or more busy CPUs";
```

6.3.8.2 The `rate` Operator

The `rate` operator computes the rate of change of an arithmetic expression as shown in the following example:

```
rate mem.freemem
```

It returns the rate of change for the `mem.freemem` performance metric; that is, the rate at which free physical memory is being allocated or released.

The `rate` intrinsic operator is most useful for metrics with instantaneous value semantics. For metrics with counter semantics, `pmie` already performs an implicit rate calculation (see the Section 6.3.4, page 130) and the `rate` operator would produce the second derivative with respect to time, which is less likely to be useful.

6.3.8.3 Transitional Operators

In some cases, an action needs to be triggered when an expression changes from true to false or vice versa. The following operators take a logical expression as an operand, and return a logical expression:

`rising`	Has the value `true` when the operand transitions from `false` to `true` in consecutive samples.
`falling`	Has the value `false` when the operand transitions from `true` to `false` in consecutive samples.

6.4 `pmie` Examples

The examples presented in this section are task-oriented and use the full power of the `pmie` specification language as described in Section 6.3, page 124.

Source code for the `pmie` examples in this chapter, and many more examples, is provided in the PCP subsystem `pcp.sw.demo`, and when installed may be found in `/var/pcp/demos/pmie`. Example 5, page 139, and Example 6, page 140, illustrate monitoring CPU utilization and disk activity.

Example 5: Monitoring CPU Utilization

```
// Some Common Performance Monitoring Scenarios
//
// The CPU Group
//
delta = 2 sec;  // more often for demonstration purposes
// common prefixes
//
percpu  = "kernel.percpu";
all     = "kernel.all";
// Unusual usr-sys split when some CPU is more than 20% in usr mode
// and sys mode is at least 1.5 times usr mode
//
cpu_usr_sys =
       some_inst (
           $percpu.cpu.sys > $percpu.cpu.user * 1.5 &&
           $percpu.cpu.user > 0.2
       )
           ->  alarm "Unusual sys time: " "%i ";
// Over all CPUs, syscall_rate > 1000 * no_of_cpus
//
cpu_syscall =
       $all.syscall > 1000 count/sec * hinv.ncpu
       ->  print "high aggregate syscalls: %v";
// Sustained high syscall rate on a single CPU
```

```
//
delta = 30 sec;
percpu_syscall =
       some_inst (
           $percpu.syscall > 2000 count/sec
       )
           -> syslog "Sustained syscalls per second? " "[%i] %v ";
// the 1 minute load average exceeds 5 * number of CPUs on any host
hosts = ":gonzo :moomba";   // change as required
delta = 1 minute;           // no need to evaluate more often than this
high_load =
     some_host (
           $all.load $hosts #'1 minute' > 5 * hinv.ncpu
       )
           -> alarm "High Load Average? " "%h: %v ";
```

Example 6: Monitoring Disk Activity

```
// Some Common Performance Monitoring Scenarios
//
// The Disk Group
//
delta = 15 sec;         // often enough for disks?
// common prefixes
//
disk    = "disk";
// Any disk performing more than 40 I/Os per second, sustained over
// at least 30 seconds is probably busy
//
delta = 30 seconds;
disk_busy =
       some_inst (
           $disk.dev.total > 40 count/sec
       )
]      -> shell "Mail -s 'Heavy systained disk traffic' sysadm";
// Try and catch bursts of activity ... more than 60 I/Os per second
// for at least 25% of 8 consecutive 3 second samples
//
delta = 3 sec;
disk_burst =
       some_inst (
           25%_sample (
               $disk.dev.total @0..7 > 60 count/sec
```

```
            )
        )
        -> alarm "Disk Burst? " "%i ";
// any SCSI disk controller performing more than 3 Mbytes per
// second is busy
// Note: the obscure 512 is to convert blocks/sec to byte/sec,
//       and pmie handles the rest of the scale conversion
//
some_inst $disk.ctl.blktotal * 512 > 3 Mbyte/sec
           -> alarm "Busy Disk Controller: " "%i ";
```

6.5 Developing and Debugging pmie Rules

Given the -d command line option, pmie executes in interactive mode, and the user is presented with a menu of options:

```
pmie debugger commands
     f [file-name]      - load expressions from given file or stdin
     l [expr-name]      - list named expression or all expressions
     r [interval]       - run for given or default interval
     S time-spec        - set start time for run
     T time-spec        - set default interval for run command
     v [expr-name]      - print subexpression for %h, %i and %v bindings
     h or ?             - print this menu of commands
     q                  - quit
pmie>
```

If both the -d option and a filename are present, the expressions in the given file are loaded before entering interactive mode. Interactive mode is useful for debugging new rules.

6.6 Caveats and Notes on pmie

The following sections provide important information for users of pmie.

6.6.1 Performance Metrics Wraparound

Performance metrics that are cumulative counters may occasionally overflow their range and wraparound to 0. When this happens, an unknown value (printed as ?) is returned as the value of the metric for one sample (recall that the value returned is normally a rate). You can have PCP interpolate a value

based on expected rate of change by setting the PCP_COUNTER_WRAP environment variable.

6.6.2 pmie Sample Intervals

The sample interval (*delta*) should always be long enough, particularly in the case of rates, to ensure that a meaningful value is computed. Interval may vary according to the metric and your needs. A reasonable minimum is in the range of ten seconds or several minutes. Although the PMCS supports sampling rates up to hundreds of times per second, using small sample intervals creates unnecessary load on the monitored system.

6.6.3 pmie Instance Names

When you specify a metric instance name (#*identifier*) in a pmie expression, it is compared against the instance name supplied by the PMCS as follows:

- If the given instance name and the PMCS name are the same, they are considered to match.
- Otherwise, the first two space separated tokens are extracted from the PMCS name. If the given instance name is the same as either of these tokens, they are considered a match.

For some metrics, notably the per process (proc.xxx.xxx) metrics, the first token in the PMCS instance name is impossible to determine at the time you are writing pmie expressions. The above policy circumvents this problem.

6.6.4 pmie Error Detection

The parser used in pmie is currently not robust in handling syntax errors. It is suggested that you check any problematic expressions individually in interactive mode:

```
pmie -v -d
pmie> f
expression
Ctrl+D
```

If the expression was parsed, its internal representation is shown:

```
pmie> l
```

The expression is evaluated twice and its value printed:

```
pmie> r 10sec
```

Then quit:

```
pmie> q
```

It is not always possible to detect semantic errors at parse time. This happens when a performance metric descriptor is not available from the named host at this time. A warning is issued, and the expression is put on a wait list. The wait list is checked periodically (about every five minutes) to see if the metric descriptor has become available. If an error is detected at this time, a message is printed to the standard error stream (`stderr`) and the offending expression is put aside.

6.7 Creating `pmie` Rules with `pmieconf`

The `pmieconf` tool is a command line utility that is designed to aid the specification of `pmie` rules from parameterized versions of the rules. `pmieconf` is used to displaying and modify variables or parameters controlling the details of the generated `pmie` rules.

`pmieconf` reads two different forms of supplied input files and produces a localized `pmie` configuration file as its output.

The first input form is a generalized `pmie` rule file such as those found below `/var/pcp/config/pmieconf/*/*`. These files contain the generalized rules which `pmieconf` is able to manipulate. Each of the rules can be enabled or disabled, or the individual variables associated with each rule can be edited.

The second form is an actual `pmie` configuration file (that is, a file which can be interpreted by `pmie`, conforming to the `pmie` syntax described in Section 6.3, page 124). This file is both input to and output from `pmieconf`.

The input version of the file contains any changed variables or rule states from previous invocations of `pmieconf`, and the output version contains both the changes in state (for any subsequent `pmieconf` sessions) and the generated `pmie` syntax. The `pmieconf` state is embedded within a `pmie` comment block at the head of the output file and is not interpreted by `pmie` itself.

`pmieconf` is an integral part of the `pmie` daemon management process described Section 6.9, page 150. Procedure 1, page 144, and Procedure 2, page 144, introduce the `pmieconf` tool through a series of typical operations.

Procedure 1: Display pmieconf Rules

1. Start pmieconf interactively.

```
$ pmieconf -f /tmp/pmiefile
Updates will be made to /tmp/pmiefile

pmieconf>
```

2. List the set of available `pmieconf` rules by using the `rules` command.
3. List the set of rule groups using the `groups` command.
4. List only the enabled rules, using the `rules enabled` command.
5. List a single rule:

```
pmieconf> list memory.swap_low
   rule: memory.swap_low  [Low free swap space]
   help: There is only threshold percent swap space remaining - the system
         may soon run out of virtual memory.  Reduce the number and size of
         the running programs or add more swap(1) space before it
completely
         runs out.
         predicate =
           some_host (
               ( 100 * ( swap.free $hosts$ / swap.length $hosts$ ) )
                 < $threshold$
               && swap.length $hosts$ > 0        // ensure swap in use
            )
   vars: enabled = no
         threshold = 10%

pmieconf>
```

6. List one rule variable:

```
pmieconf> list memory.swap_low threshold
   rule: memory.swap_low  [Low free swap space]
         threshold = 10%

pmieconf>
```

Procedure 2: Modify pmieconf Rules and Generate a pmie File

1. Lower the threshold for the `memory.swap_low` rule, and also change the `pmie` sample interval affecting just this rule. The *delta* variable is special in

that it is not associated with any particular rule; it has been defined as a global `pmieconf` variable. Global variables can be displayed using the `list global` command to `pmieconf`, and can be modified either globally or local to a specific rule.

```
pmieconf> modify memory.swap_low threshold 5

pmieconf> modify memory.swap_low delta "1 sec"

pmieconf>
```

2. Disable all of the rules except for the `memory.swap_low` rule so that you can see the effects of your change in isolation.

   This produces a relatively simple `pmie` configuration file:

```
pmieconf> disable all

pmieconf> enable memory.swap_low

pmieconf> status
  verbose:  off
  enabled rules:  1 of 35
  pmie configuration file:  /tmp/pmiefile
  pmie processes (PIDs) using this file:  (none found)

pmieconf> quit
```

   You can also use the `status` command to verify that only one rule is enabled at the end of this step.

3. Run `pmie` with the new configuration file. Use a text editor to view the newly generated `pmie` configuration file (`/tmp/pmiefile`), and then run the command:

```
$ pmie -T "1.5 sec" -v -l /tmp/log /tmp/pmiefile
memory.swap_low: false

memory.swap_low: false

$ cat /tmp/log
Log for pmie on moomba started Mon Jun 21 16:26:06 1999

pmie: PID = 21847, default host = moomba
```

```
[Mon Jun 21 16:26:07] pmie(21847) Info: evaluator exiting

Log finished Mon Jun 21 16:26:07 1999
$
```

4. Notice that both of the `pmieconf` files used in the previous step are simple text files, as described in the `pmieconf`(4) man page:

```
$ file /tmp/pmiefile
/tmp/pmiefile:  PCP pmie config (V.1)
$ file /var/pcp/config/pmieconf/memory/swap_low
/var/pcp/config/pmieconf/memory/swap_low:       PCP pmieconf rules (V.1)
```

6.8 Creating `pmie` Rules with `pmrules`

The GUI tool `pmrules` may be used to generate `pmie` rules from templates that are shipped with PCP. These templates are parameterized versions of rules describing common performance scenarios suited for `pmie` monitoring.

Procedure 3: Creating `pmie` Rules

1. Start `pmrules`, and choose Import... from the Template menu.
2. Click the `Choose File...` button in the "Import template(s) from file" dialog.

   Sample templates are installed in the directory `/var/pcp/config/pmrules`.
3. Double-click the `pcp` directory in the `pmrules` directory browser window.

   An `Import template(s) from file` dialog appears, as shown in Figure 31.

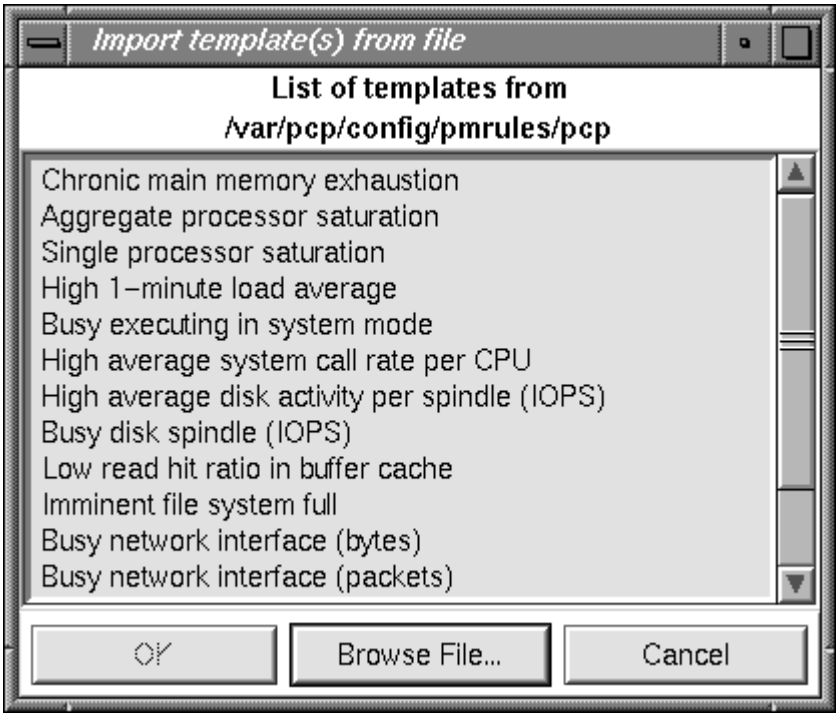

Figure 31. `pmrules` `Import template(s) from file` Dialog

4. Select the desired templates, click OK, and return to the `pmrules` main window, which appears similar to the one shown in Figure 32.

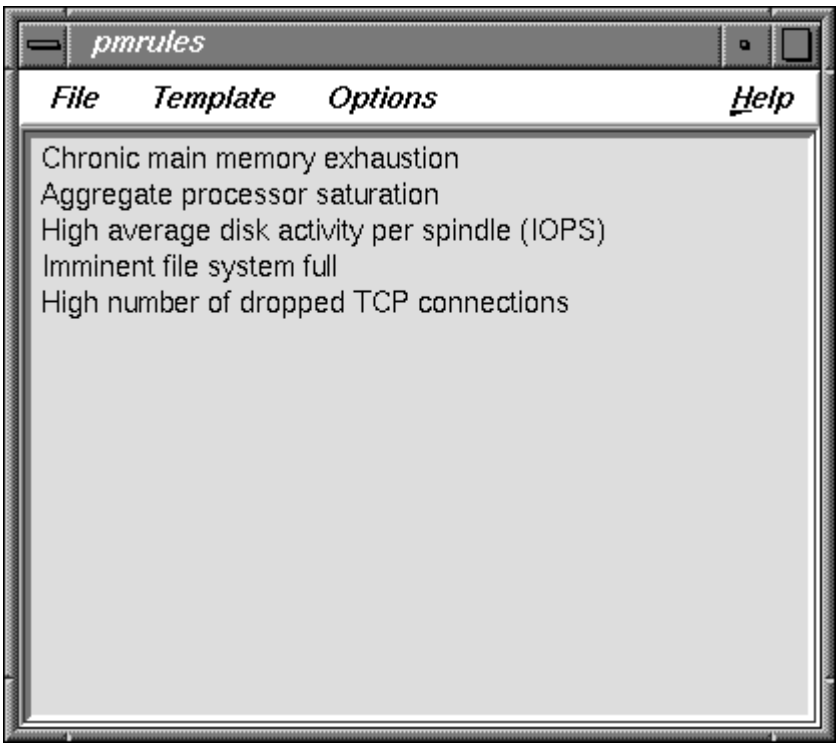

Figure 32. `pmrules` Main Dialog after Template Selection

5. Double-click the desired template, and the `pmrules` `Edit template` dialog displays, similar to the one shown in Figure 33.

   At this point, you can customize the template by assigning values to the `threshold`, `delta`, and `holdoff` `Parameters` text boxes, then either selecting one of the predefined `Actions`, or specifying your own custom user action.

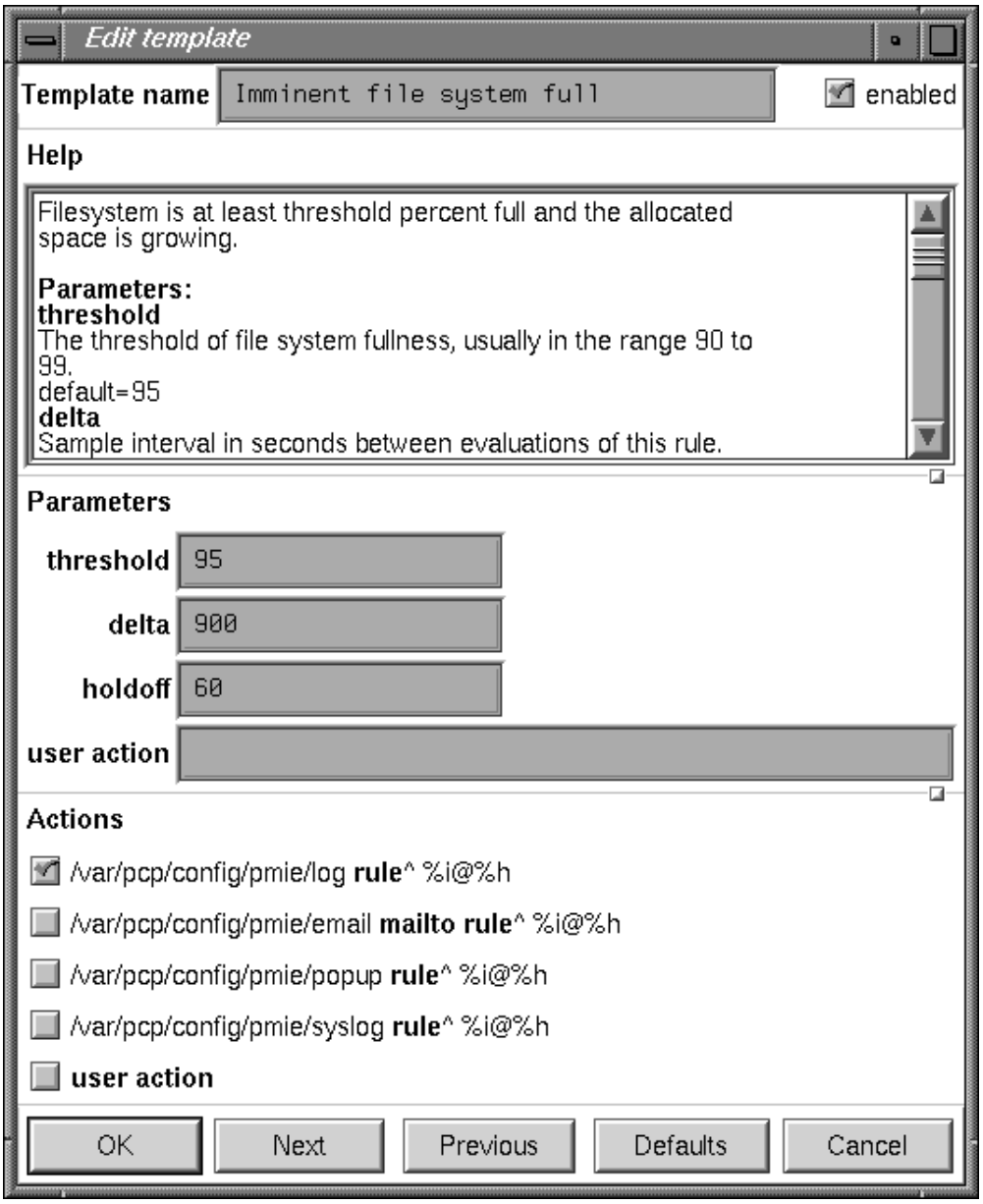

Figure 33. `pmrules Edit template` Dialog

6. When you are finished customizing the template, click OK and return to the main `pmrules` window.
7. Choose `Save As` from the `File` menu, and provide a new name for your private copy of the `pmrules` template file.

   Two files are saved. The first one takes the given filename and is your private copy of the `pmrules` template file. The second file takes the given filename with the suffix `.pmie` appended and contains the `pmie` rules—this second file should be given as an argument to `pmie`.

You can also create new templates for other performance problems. These can then be included in the template collection available to `pmrules`, and then used to customize instances of the `pmie` rules for particular hosts.

See the `pmrules`(1) man page for a complete description of the capabilities of the `pmrules` tool.

6.9 Management of `pmie` Processes

The `pmie` process can be run as a daemon as part of the system startup sequence, and can thus be used to perform automated, live performance monitoring of a running system. To do this, run these commands (as superuser):

```
# chkconfig pmie on
# /etc/init.d/pmie start
```

By default, these enable a single `pmie` process monitoring the local host, with the default set of `pmieconf` rules enabled (for more information about `pmieconf`, see Section 6.7, page 143). Procedure 4, page 150, illustrates how you can use these commands to start any number of `pmie` processes to monitor local or remote machines.

Procedure 4: Add a New `pmie` Instance to the `pmie` Daemon Management Framework

1. Use a text editor (as superuser) to edit the `pmie` control file `/var/pcp/config/pmie/control`. Notice the default entry toward the end of the file, which looks like this:

```
#Host           S?  Log File                                  Arguments
LOCALHOSTNAME   n   /var/adm/pmielog/LOCALHOSTNAME/pmie.log   -c config.default
```

This entry is used to enable a local pmie process. Add a new entry for a remote host on your local network (for example, moomba), by using your pmie configuration file (see Section 6.7, page 143):

```
#Host           S?  Log File                                  Arguments
moomba          n   /var/adm/pmielog/moomba/pmie.log          -c /tmp/pmiefile
```

2. Enable pmie daemon management:

   ```
   # chkconfig pmie on
   ```

   This simple step allows pmie to be started as part of your machine's boot process.

3. Start the two pmie daemons. At the end of this step, you should see two new pmie processes monitoring the local and remote hosts:

   ```
   # /etc/init.d/pmie start
       Performance Co-Pilot starting inference engine(s) ...
   ```

   Wait a few moments while the startup scripts run. The pmie start script uses the pmie_check script to do most of its work.

   Verify that the pmie processes have started using the pmie metrics exported by the pmcd PMDA (wobbly is the local host):

```
# pminfo -f pmcd.pmie.pmcd_host

pmcd.pmie.pmcd_host
    inst [23150 or "23150"] value "wobbly.melbourne.sgi.com"
    inst [23204 or "23204"] value "moomba.melbourne.sgi.com"
```

If a remote host is not up at the time when pmie is started, the pmie process may exit. pmie processes may also exit if the local machine is starved of memory resources. To counter these adverse cases, it can be useful to have a crontab entry running. Adding an entry as shown in Procedure 5, page 152, ensures that if one of the configured pmie processes exits, it is automatically restarted.

Procedure 5: Add a `pmie` `crontab` Entry

1. Merge the sample `pmie` `crontab` entry with your `root` `crontab` entry. The `/var/pcp/config/pmie/crontab` file holds this sample entry:

```
$ cat /var/pcp/config/pmie/crontab
#
# standard Performance Co-Pilot crontab entries for a PCP site
# with one or more pmie instances running
#
# every 30 minutes, check pmie instances are running
25,55   *       *       *       *       /usr/pcp/bin/pmie_check
```

2. Use the `crontab` command and a text editor to append the sample `pmie` `crontab` entry to `root` `crontab` file. This procedure runs the `pmie_check` script once every thirty minutes to verify that the `pmie` instances are running. If they are not, the procedure restarts them and sends e-mail to `root` indicating which instances needed restarting.

6.9.1 Global Files and Directories

The following global files and directories influence the behavior of `pmie` and the `pmie` management scripts:

`/etc/config/pmie`

Controls the `pmie` daemon facility. Enable it using this command:

```
chkconfig pmie on
```

`/var/pcp/demos/pmie/*`

Contains sample `pmie` rules that may be used as a basis for developing local rules.

`/var/pcp/config/pmie/config.default`

Is the default `pmie` configuration file that is used when the `pmie` daemon facility is enabled.

`/var/pcp/config/pmieconf/*/*`

Contains the `pmieconf` rule definitions in its subdirectories.

`/var/pcp/config/pmie/control`

Defines which PCP collector hosts require a daemon `pmie` to be launched on the local host, where the configuration file comes from, where the `pmie` log file should be created, and `pmie` startup options.

`/var/pcp/config/pmlogger/crontab`

Contains prototype `crontab` entries that may be merged with the `crontab` entries for root to schedule the periodic execution of the `pmie_check` script, for verifying that `pmie` instances are running.

`/var/adm/pmielog/`

Contains the `pmie` log files for the host. These files are created by the default behavior of the `/etc/init.d/pmie` startup scripts.

6.9.2 `pmie` Instances and Their Progress

The `pmcd` PMDA exports information about executing `pmie` instances and their progress in terms of rule evaluations and action execution rates.

`pmie_check`	This command is similar to the `pmlogger` support script, `pmlogger_check`.
`/etc/init.d/pmie`	This control file supports the starting and stopping of multiple `pmie` instances that are monitoring one or more hosts.
`/var/tmp/pmie`	The statistics that `pmie` gathers are maintained in binary data structure files. These files are in the `/var/tmp/pmie` directory.
`pmcd.pmie` metrics	If `pmie` is running on a system with a PCP collector deployment, the `pmcd` PMDA exports these metrics via the `pmcd.pmie` group of metrics.

Archive Logging [7]

Performance monitoring and management in complex systems demands the ability to accurately capture performance characteristics for subsequent review, analysis, and comparison. Performance Co-Pilot (PCP) provides extensive support for the creation and management of archive logs that capture a user-specified profile of performance information to support retrospective performance analysis.

The following major sections are included in this chapter:

- Section 7.1, page 155, presents the concepts and issues involved with creating and using archive logs.
- Section 7.2, page 158, describes the interaction of the PCP tools with archive logs.
- Section 7.3, page 163, shows some shortcuts for setting up useful PCP archive logs.
- Section 7.4, page 167, provides information about other archive logging features and sevices.
- Section 7.5, page 170, presents helpful directions if your archive logging implementation is not functioning correctly.

7.1 Introduction to Archive Logging

Within the Performance Co-Pilot, the `pmlogger` utility may be configured to collect archives of performance metrics. The archive creation process is easy and very flexible, incorporating the following features:

- Archive log creation at either a PCP collector (typically a server) or a PCP monitor system (typically a workstation), or at some designated PCP archive logger host.
- Concurrent independent logging, both local and remote. The performance analyst can activate a private `pmlogger` instance to collect only the metrics of interest for the problem at hand, independent of other logging on the workstation or remote host.
- Record mode in various GUI monitoring tools to create archives as needed from the current visualization.

- Independent determination of logging frequency for individual metrics or metric instances. For example, you could log the "5 minute" load average every half hour, the write I/O rate on the DBMS log spindle every 10 seconds, and aggregate I/O rates on the other disks every minute.
- Dynamic adjustment of what is to be logged, and how frequently, via `pmlc`. This feature may be used to disable logging or to increase the sample interval during periods of low activity or chronic high activity (to minimize logging overhead and intrusion). A local `pmlc` may interrogate and control a remote `pmlogger`, subject to the access control restrictions implemented by `pmlogger`.
- Self-contained logs that include all system configuration and metadata required to interpret the values in the log. These logs can be kept for analysis at a much later time, potentially after the hardware or software has been reconfigured and the logs have been stored as discrete, autonomous files for remote analysis.
- `cron`-based scripts to expedite the operational management, for example, log rotation, consolidation, and culling.
- Archive folios as a convenient aggregation of multiple archive logs. Archive folios may be created with the `mkafm` utility and processed with the `pmafm` tool.

7.1.1 Archive Logs and the PMAPI

Critical to the success of the PCP archive logging scheme is the fact that the library routines providing access to real-time feeds of performance metrics also provide access to the archive logs.

Live feeds (or real-time) sources of performance metrics and archives are literally interchangeable, with a single Performance Metrics Application Programming Interface (PMAPI) that preserves the same semantics for both styles of metric source. In this way, applications and tools developed against the PMAPI can automatically process either live or historical performance data.

The only restriction is that both live and historical data cannot be monitored simultaneously with the same invocation of a visualization tool.

7.1.2 Retrospective Analysis Using Archive Logs

One of the most important applications of archive logging services provided by PCP is in the area of retrospective analysis. In many cases, understanding

today's performance problems can be assisted by side-by-side comparisons with yesterday's performance. With routine creation of performance archive logs, you can concurrently replay pictures of system performance for two or more periods in the past.

Archive logs are also an invaluable source of intelligence when trying to diagnose what went wrong, as in a performance postmortem. Because the PCP archive logs are entirely self-contained, this analysis can be performed off-site if necessary.

Each archive log contains metric values from only one host. However, many PCP tools can simultaneously visualize values from multiple archives collected from different hosts.

The archives can be replayed against the inference engine (`pmie` is an application that uses the PMAPI). This allows you to automate the regular, first-level analysis of system performance.

Such analysis can be performed by constructing suitable expressions to capture the essence of common resource saturation problems, then periodically creating an archive and playing it against the expressions. For example, you may wish to create a daily performance audit (run by the `cron` command) to detect performance regressions.

For more about `pmie`, see Chapter 6.

7.1.3 Snapshots from PCP Archive Logs

Periodic snapshot images of recent performance, activity levels, and resource utilization can be extracted from the PCP archive logs and published via a World Wide Web (WWW) server. These are high-quality images generated from `pmchart` that provide an excellent vehicle for publishing performance summary information for users, system and network administrators, or managers. The `pmsnap` services may be used to automate snapshots. For additional information, see the `pmsnap`(1) man page.

7.1.4 Using Archive Logs for Capacity Planning

By collecting performance archives with relatively long sampling periods, or by reducing the daily archives to produce summary logs, the capacity planner can collect the base data required for forward projections, and can estimate resource demands and explore "what if" scenarios by replaying data using visualization tools and the inference engine.

7.2 Using Archive Logs with Performance Visualization Tools

Most PCP tools default to real-time display of current values for performance metrics from PCP collector host(s). However, most PCP tools also have the capability to display values for performance metrics retrieved from PCP archive log(s). The following sections describe plans, steps, and general issues involving archive logs and the PCP tools.

7.2.1 Coordination between `pmlogger` and PCP tools

Most commonly, a PCP tool would be invoked with the `-a` option to process an archive log some time after `pmlogger` had finished creating the archive. However, a tool such as `oview` that uses a Time Control dialog (see Section 3.4) stops when the end of archive is reached, but could resume if more data is written to the PCP archive log.

> **Note:** `pmlogger` uses buffered I/O to write the archive log so that the end of the archive may be aligned with an I/O buffer boundary, rather than with a logical archive log record. If such an archive was read by a PCP tool, it would appear truncated and might confuse the tool. These problems may be avoided by sending `pmlogger` a `SIGUSR1` signal, or by using the `flush` command of `pmlc` to force `pmlogger` to flush its output buffers.

7.2.2 Administering PCP Archive Logs Using `cron` Scripts

The IRIX operating system supports the standard `cron` process scheduling system. Complete information on the `cron` command is available in the appropriate man page and in *IRIX Admin: System Configuration and Operation*.

Performance Co-Pilot supplies shell scripts to use the `cron` functionality to help manage your archive logs. The following scripts are supplied:

Script	Description
`pmlogger_daily`	Performs a daily housecleaning of archive logs and notices.
`pmlogger_merge`	Merges archive logs and is called by `pmlogger_daily`.
`pmlogger_check`	Checks to see that all desired `pmlogger` processes are running on your system, and invokes any that are missing for any reason.

`pmsnap`	Generates graphic image snapshots of `pmchart` performance charts at regular intervals.

The configuration files used by these scripts can be edited to suit your particular needs, and are generally controlled by the `/var/pcp/config/pmlogger/control` file (`pmsnap` has an additional control file). Complete information on these scripts is available in the `pmlogger_daily`(1) and `pmsnap`(1) man pages.

7.2.3 Archive Log File Management

Performance Co-Pilot archive log files can occupy a great deal of disk space, and management of archive logs can be a large task in itself. The following sections provide information to assist you in PCP archive log file management.

7.2.3.1 Basename Conventions

When a PCP archive is created by `pmlogger`, an archive basename must be specified and several physical files are created, as shown in Table 3.

Table 3. Filenames for PCP Archive Log Components (`archive`.*)

Filename	Contents
`archive.`*index*	Temporal index for rapid access to archive contents.
`archive.`*meta*	Metadata descriptions for performance metrics and instance domains appearing in the archive.
`archive.N`	Volumes of performance metrics values, for `N` = 0,1,2,...

7.2.3.2 Log Volumes

A single PCP archive may be partitioned into a number of volumes. These volumes may expedite management of the archive; however, the metadata file and at least one volume must be present before a PCP tool can process the archive.

You can control the size of an archive log volume by using the `-v` command line option to `pmlogger`. This option specifies how large a volume should become before `pmlogger` starts a new volume. Archive log volumes retain the same base filename as other files in the archive log, and are differentiated by a

numeric suffix that is incremented with each volume change. For example, you might have a log volume sequence that looks like this:

```
netserver.log.0
netserver.log.1
netserver.log.2
```

You can also cause an existing log to be closed and a new one to be opened by sending a `SIGHUP` signal to `pmlogger`, or by using the `pmlc` command to change the `pmlogger` instructions dynamically, without interrupting `pmlogger` operation. Complete information on log volumes is found in the `pmlogger`(1) man page.

7.2.3.3 Basenames for Managed Archive Log Files

The PCP archive management tools support a consistent scheme for selecting the basenames for the files in a collection of archives and for mapping these files to a suitable directory hierarchy.

Once configured, the PCP tools that manage archive logs employ a consistent scheme for selecting the basename for an archive each time `pmlogger` is launched, namely the current date and time in the format YYYYMMDD.HH.MM. Typically, at the end of each day, all archives for a particular host on that day would be merged to produce a single archive with a basename constructed from the date, namely YYYYMMDD. The `pmlogger_daily` script performs this action and a number of other routine housekeeping chores.

7.2.3.4 Directory Organization for Archive Log Files

If you are using a deployment of PCP tools and daemons to collect metrics from a variety of hosts and storing them all at a central location, you should develop an organized strategy for storing and naming your log files.

> **Note:** There are many possible configurations of `pmlogger`, as described in Section 8.3. The directory organization described in this section is recommended for any system on which `pmlogger` is configured for permanent execution (as opposed to short-term executions, for example, as launched from `pmchart` to record some performance data of current interest).

Typically, the IRIX filesystem structure can be used to reflect the number of hosts for which a `pmlogger` instance is expected to be running locally, obviating the need for lengthy and cumbersome filenames. It makes considerable sense to place all logs for a particular host in a separate directory

named after that host. Because each instance of `pmlogger` can only log metrics fetched from a single host, this also simplifies some of the archive log management and administration tasks.

For example, consider the filesystem and naming structure shown in Figure 34.

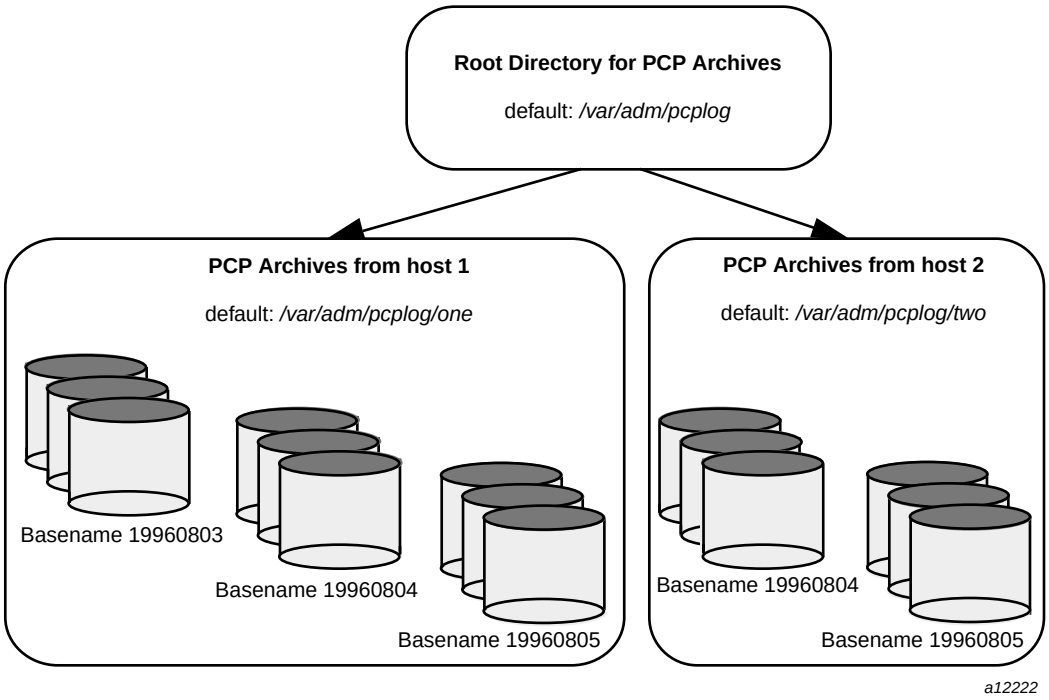

Figure 34. Archive Log Directory Structure

The specification of where to place the archive log files for particular `pmlogger` instances is encoded in the configuration file `/var/pcp/config/pmlogger/control`, and this file should be customized on each host running an instance of `pmlogger`.

If many archives are being created, and the associated PCP collector systems form peer classes based upon service type (for example, Web servers, DBMS servers, NFS servers, and so on), then it may be appropriate to introduce another layer into the directory structure, or use symbolic links to group together hosts providing similar service types.

7.2.3.5 Configuration of pmlogger

The configuration files used by pmlogger describe which metrics are to be logged. Groups of metrics may be logged at different intervals to other groups of metrics. Two states, mandatory and advisory, also apply to each group of metrics, defining whether metrics definitely should be logged or not logged, or whether a later advisory definition may change that state.

The mandatory state takes precedence if it is on or off, causing any subsequent request for a change in advisory state to have no effect. If the mandatory state is maybe, then the advisory state determines if login is enabled or not.

The mandatory states are on, off, and maybe. The advisory states, which only affect metrics that are mandatory maybe, are on and off. Therefore, a metric that is mandatory maybe in one definition and advisory on in another definition would be logged at the advisory interval. Metrics that are not specified in the pmlogger configuration file are mandatory maybe and advisory off by default and are not logged.

A complete description of the pmlogger configuration format can be found on the pmlogger(1) man page.

7.2.3.6 PCP Archive Contents

Once a PCP archive log has been created, the pmdumplog utility may be used to display various information about the contents of the archive. For example, start with the following command:

```
pmdumplog -l /var/adm/pcplog/www.sgi.com/960731
```

It might produce the following output:

```
Log Label (Log Format Version 1)
Performance metrics from host www.sgi.com
     commencing Wed Jul 31 00:16:34.941 1996
     ending     Thu Aug  1 00:18:01.468 1996
```

The simplest way to discover what performance metrics are contained within an archive is to use pminfo; for example:

```
pminfo -a /var/adm/pcplog/www.sgi.com/960731 network.mbuf
network.mbuf.alloc
network.mbuf.typealloc
network.mbuf.clustalloc
network.mbuf.clustfree
network.mbuf.failed
```

```
network.mbuf.waited
network.mbuf.drained
```

7.3 Cookbook for Archive Logging

The following sections present a checklist of tasks that may be performed to enable PCP archive logging with minimal effort. For a complete explanation, refer to the other sections in this chapter and the man pages for `pmlogger` and related tools.

7.3.1 Primary Logger

Assume you wish to activate primary archive logging on the PCP collector host `pluto`. Execute all of the following tasks while logged into `pluto` as the superuser (`root`).

1. Create the directory to hold the archive logs:

   ```
   mkdir /var/adm/pcplog/pluto
   ```

2. Choose a suitable `pmlogger` configuration file. Here are some examples:

   - The default configuration: `/var/pcp/config/pmlogger/config.default.`
   - A broad summary configuration, sufficient to be used with `dkvis`, `mpvis`, `nfsvis`, and `pmkstat`: `/var/pcp/config/pmlogger/config.Summary.`
   - One of the other `config.*` files in the `/var/pcp/config/pmlogger` directory, tailored for an application, a PCP add-on product, a `pmchart` view, or a PCP monitor tool.

     Copy the chosen configuration file to `/var/adm/pcplog/pluto/config.default` (possibly after some customization).

3. Edit `/var/pcp/config/pmlogger/control`. Using the line for the "local primary logger" as a template, add the following line to the file:

   ```
   pluto  y  n  /var/adm/pcplog/pluto  -c config.keep
   ```

4. Make sure `pmcd` and `pmlogger` are enabled and running:

```
chkconfig pmcd on
chkconfig pmlogger on
/etc/init.d/pcp start
Performance Co-Pilot PMCD started (logfile is .... /pmcd.log)
Performance Co-Pilot Primary Logger started
```

5. Verify that the primary `pmlogger` instance is running:

```
pmlc
pmlc> connect primary
pmlc> status
pmlogger [primary] on host pluto is logging metrics from host pluto
log started      Thu Aug  8 14:33:01 1996 (times in local time)
last log entry   Thu Aug  8 14:34:11 1996
current time     Thu Aug  8 14:36:54 1996
log volume       0
log size         284
```

6. Verify that the archive files are being created in the correct place:

```
ls /var/adm/pcplog/pluto
960808.14.33.0
960808.14.33.index
960808.14.33.meta
Latest
pmlogger.log
```

7.3.2 Other Logger Configurations

Assume you wish to create archive logs on the local host for performance metrics collected from the remote host `bert`. Execute all of the following tasks while logged into the local host as the superuser (`root`).

1. Create the directory to hold the archive logs:

```
mkdir /var/adm/pcplog/bert
```

2. Choose a suitable `pmlogger` configuration file. Here are three examples:

   - The default configuration: `/var/pcp/config/pmlogger/config.default.`
   - A broad summary configuration, sufficient to be used with `dkvis`, `mpvis`, `nfsvis`, and `pmkstat`: `/var/pcp/config/pmlogger/config.Summary.`

- One of the other `config.*` files in the `/var/pcp/config/pmlogger` directory, tailored for an application, a PCP add-on product, a `pmchart` view, or a PCP monitor tool.

  Copy the chosen configuration file to `/var/adm/pcplog/bert/config.default` (possibly after some customization).

3. Edit `/var/pcp/config/pmlogger/control.` Using the line for `remote` as a template, add the following line to the file:

```
bert  n  n  /var/adm/pcplog/bert  -c ./config.default
```

4. Start `pmlogger`:

```
/usr/pcp/bin/pmlogger_check
Restarting pmlogger for host "bert" ..... done
```

5. Verify that the `pmlogger` instance is running:

```
pmlc
pmlc> show loggers
The following pmloggers are running on bert:
    primary (19144)
pmlc> connect 19144
pmlc> status
pmlogger [19144] on host ernie is logging metrics from host bert
log started      Thu Aug  8 10:10:10 1996 (times in local time)
last log entry   Thu Aug  8 14:50:54 1996
current time     Thu Aug  8 14:55:48 1996
log volume       0
log size         256
```

7.3.3 Archive Log Administration

Assume the local host has been set up to create archive logs of performance metrics collected from one or more hosts (which may be either the local host or a remote host).

To activate the maintenance and housekeeping scripts for a collection of archive logs, execute the following tasks while logged into the local host as the superuser (`root`):

1. Augment the `crontab` file for `root`. For example:

```
crontab -l >/tmp/foo
```

2. Edit /tmp/foo, adding lines similar to those from /var/pcp/config/pmlogger/crontab for pmlogger_daily and pmlogger_check; for example:

```
# daily processing of archive logs
10     0     *     *     *       /usr/pcp/bin/pmlogger_daily
# every 30 minutes, check pmlogger instances are running
25,55  *     *     *     *       /usr/pcp/bin/pmlogger_check
```

3. Make these changes permanent with this command:

```
crontab </tmp/foo
```

7.3.4 Making Snapshot Images from Archive Logs

You may also choose to enable periodic snapshot images of performance data to be produced from the archive logs using the facilities of pmsnap; instructions for this operation can be found in Section 4.1.9, page 79 and in the man page for pmsnap(1).

Assume the local host has been set up to create archive logs of performance metrics collected from the host oscar (which may be either the local host or a remote host). Execute all of the following tasks while logged into the local host as the superuser (root).

1. Make sure the optional subsystem pcp.sw.monitor has been installed.
2. Use the /var/pcp/config/pmsnap/Summary snapshot as an example (you may wish to customize this later).
3. Ensure that the pmlogger that is collecting performance metrics from the host oscar includes all of the metrics named in the /var/pcp/config/pmlogger/config.Summary configuration file (you may wish to simply use this as the configuration file for this pmlogger instance). If necessary, reconfigure this pmlogger instance as follows:

   kill -INT *PID-of-pmlogger-instance*

   Edit the configuration file as required. Restart pmlogger with this command:

   /usr/pcp/bin/pmlogger_check

4. Check the two Summary lines in the /var/pcp/config/pmsnap/control file. You must replace LOCATHOSTNAME with oscar in both lines (unless oscar is the local host,

in which case the change is optional), and you may wish to change the directory for the output files (the default is `/var/www/htdocs/snapshots`).

5. Augment the `crontab` file for `root` to allow `pmsnap` to be run periodically. For example:

   ```
   crontab -l >/tmp/foo
   ```

6. Edit `/tmp/foo`, adding lines similar to those from `/var/pcp/config/pmlogger/crontab` for `pmsnap`; for example:

   ```
   # every 30 minutes, generate performance snapshot images
   30,0   *     *     *     *       /usr/pcp/bin/pmsnap -d :0
   ```

   The snapshots are produced using `pmchart`, and this tool requires connection to an X server. If the local host is not running an X server, then you must locate a system with an active X server, and ensure that this X server will accept connections from remote X clients; see the `xhost`(1) man page for details. If this host is `grover`, then replace `-d :0` in the line above with `-d grover:0`

   Other options for gaining access to an active X server are discussed in the `pmsnap`(1) man page.

7. Make these changes permanent with this command:

   ```
   crontab </tmp/foo
   ```

8. After 30 minutes or so (time enough for the `cron` command to complete), check that the GIF files have been created:

   ```
   ls -l /var/www/htdocs/snapshots
   ```

9. Create a Web page that includes the images. A sample file of HTML source is provided in `/var/pcp/config/pmsnap/Summary.html`.

7.4 Other Archive Logging Features and Services

Other archive logging features and services include PCP archive folios, manipulating archive logs, primary logger, and using `pmlc`.

7.4.1 PCP Archive Folios

A collection of one or more PCP archive logs may be combined with a control file to produce a PCP archive folio. Archive folios are created using either `mkaf` or the interactive record mode services of various PCP GUI monitoring tools.

The automated archive log management services also create an archive folio named `Latest` for each managed `pmlogger` instance, to provide a symbolic name to the most recent archive log. With reference to Figure 34, this would mean the creation of the folios `/var/adm/pcplog/one/Latest` and `/var/adm/pcplog/two/Latest`.

The `pmafm` utility is completely described in the `pmafm`(1) man page, and provides the interactive commands (single commands may also be executed from the command line) for the following services:

- Checking the integrity of the archives in the folio.
- Displaying information about the component archives.
- Executing PCP tools with their source of performance metrics assigned concurrently to all of the component archives (where the tool supports this), or serially executing the PCP tool once per component archive.
- If the folio was created by a single PCP monitoring tool, replaying all of the archives in the folio with that monitoring tool.
- Restricting the processing to particular archives, or the archives associated with particular hosts.

7.4.2 Manipulating Archive Logs with `pmlogextract`

The `pmlogextract` tool takes a number of PCP archive logs from a single host and performs the following tasks:

- Merges the archives into a single log, while maintaining the correct time stamps for all values
- Extracts all metric values within a temporal window that could encompass several archive logs
- Extracts only a configurable subset of metrics from the archive logs

See the `pmlogextract`(1) man page for full information on this command. It replaces functionality of the `pmlogmerge` tool as of PCP release 2.0.

7.4.3 Primary Logger

On each system for which `pmcd` is active (each PCP collector system), there is an option to have a distinguished instance of the archive logger `pmlogger` (the "primary" logger) launched each time `pmcd` is started. This may be used to ensure the creation of minimalist archive logs required for ongoing system management and capacity planning in the event of failure of a system where a remote `pmlogger` may be running, or because the preferred archive logger deployment is to activate `pmlogger` on each PCP collector system.

Run the following command as superuser on each PCP collector system where you want to activate the primary `pmlogger`:

```
chkconfig pmlogger on
```

The primary logger launches the next time `pmcd` is started. If you wish this to happen immediately, follow up with this command:

```
/etc/init.d/pcp start
```

When it is started in this fashion, the `/etc/config/pmlogger.options` file provides command line options for `pmlogger`. In the default setup, this in turn means that the initial logging state and configuration is specified in the file `/var/pcp/config/pmlogger/config.default`. Either one or both of these files may be modified to tailor `pmlogger` operation to the local requirements.

7.4.4 Using `pmlc`

You may tailor `pmlogger` dynamically with the `pmlc` command. Normally, the `pmlogger` configuration is read at startup. If you choose to modify the `config` file to change the parameters under which `pmlogger` operates, you must stop and restart the program for your changes to have effect. Alternatively, you may change parameters whenever required by using the `pmlc` interface.

To run the `pmlc` tool, enter:

```
pmlc
```

By default, `pmlc` acts on the primary instance of `pmlogger` on the current host. See the `pmlc`(1) man page for a description of command line options. When it is invoked, `pmlc` presents you with a prompt:

```
pmlc>
```

You may obtain a listing of the available commands by entering a question mark (?) and pressing Enter. You see output similar to the following:

```
show loggers [@<host>]           display <pid>s of running pmloggers
connect _logger_id [@<host>]     connect to designated pmlogger
status                           information about connected pmlogger
query metric-list                show logging state of metrics
new volume                       start a new log volume
flush                            flush the log buffers to disk
log { mandatory | advisory } on <interval> _metric-list
log { mandatory | advisory } off _metric-list
log mandatory maybe _metric-list
timezone local|logger|'<timezone>' change reporting timezone
help                              print this help message
quit                              exit from pmlc
_logger_id   is  primary | <pid> | port <n>
_metric-list is  _metric-spec | { _metric-spec ... }
_metric-spec is  <metric-name> | <metric-name> [ <instance> ... ]
```

Here is an example:

```
pmlc
pmlc> show loggers @babylon
The following pmloggers are running on babylon:
       primary (1892)
pmlc> connect 1892 @babylon
pmlc> log advisory on 2 secs disk.dev.read
pmlc> query disk.dev
disk.dev.read
       adv  on  nl       5 min  [131073 or ''dks0d1'']
       adv  on  nl       5 min  [131074 or ''dks0d2'']
pmlc> quit
```

Note: Any changes to the set of logged metrics made via `pmlc` are not saved, and are lost the next time `pmlogger` is started with the same configuration file. Permanent changes are made by modifying the `pmlogger` configuration file(s).

Refer to the `pmlc`(1) and `pmlogger`(1) man pages for complete details.

7.5 Archive Logging Troubleshooting

The following issues concern the creation and use of logs using `pmlogger`.

7.5.1 pmlogger Cannot Write Log

Symptom: The pmlogger utility does not start, and you see this message:

```
_pmLogNewFile: ‘‘foo.index’’ already exists, not over-written
```

Cause: Archive logs are considered sufficiently precious that pmlogger does not empty or overwrite an existing set of archive log files. The log named foo actually consists of the physical file foo.index, foo.meta, and at least one file foo.N, where N is in the range 0, 1, 2, 3, and so on.

A message similar to the one above is produced when a new pmlogger instance encounters one of these files already in existence.

Resolution: If you are sure, remove all of the parts of the archive log. For example, use the following command:

```
rm -f foo.*
```

Then rerun pmlogger.

7.5.2 Cannot Find Log

Symptom: The pmdumplog utility, or any tool that can read an archive log, displays this message:

```
Cannot open archive mylog: No such file or directory
```

Cause: An archive consists of at least three physical files. If the base name for the archive is mylog, then the archive actually consists of the physical files mylog.index, mylog.meta, and at least one file mylog.N, where N is in the range 0, 1, 2, 3, and so on.

The above message is produced if one or more of the files is missing.

Resolution: Use this command to check which files the utility is trying to open:

```
ls mylog.*
```

Turn on the internal debug flag DBG_TRACE_LOG (-D 128) to see which files are being inspected by the _pmOpenLog routine as shown in the following example:

```
pmdumplog -D 128 -l mylog
```

Locate the missing files and move them all to the same directory, or remove all of the files that are part of the archive, and recreate the archive log.

7.5.3 Primary pmlogger Cannot Start

Symptom: The primary pmlogger cannot be started. A message like the following appears:

```
pmlogger: there is already a primary pmlogger running
```

Cause: There is either a primary pmlogger already running, or the previous primary pmlogger was terminated unexpectedly before it could perform its cleanup operations.

Resolution: If there is already a primary pmlogger running and you wish to replace it with a new pmlogger, use the show command in pmlc to determine the process ID of the primary pmlogger. The process ID of the primary pmlogger appears in parentheses after the word "primary." Send an SIGINT signal to the process to shut it down (use the kill command). If the process does not exist, proceed to the manual cleanup described in the paragraph below. If the process did exist, it should now be possible to start the new pmlogger.

If pmlc's show command displays a process ID for a process that does not exist, a pmlogger process was terminated before it could clean up. If it was the primary pmlogger, the

corresponding control files must be removed before one can start a new primary `pmlogger`. It is a good idea to clean up any spurious control files even if they are not for the primary `pmlogger`.

The control files are kept in `/var/tmp/pmlogger`. A control file with the process ID of the `pmlogger` as its name is created when the `pmlogger` is started. In addition, the primary `pmlogger` creates a symbolic link named `primary` to its control file.

For the primary `pmlogger`, remove both the symbolic link and the file (corresponding to its process ID) to which the link points. For other `pmlogger`s, remove just the process ID file. Do not remove any other files in the directory. If the control file for an active `pmlogger` is removed, `pmlc` is not able to contact it.

7.5.4 Identifying an Active `pmlogger` Process

Symptom: You have a PCP archive log that is demonstrably growing, but do not know the identify of the associated `pmlogger` process.

Cause: The PID is not obvious from the log, or the archive name may not be obvious from the output of the `ps` command.

Resolution: If the archive basename is `foo`, run the following commands:

```
pmdumplog -l foo
Log Label (Log Format Version 1)
Performance metrics from host gonzo
     commencing Wed Aug  7 00:10:09.214 1996
     ending     Wed Aug  7 16:10:09.155 1996
pminfo -a foo -f pmcd.pmlogger
pmcd.pmlogger.host
     inst [10728 or "10728"] value "gonzo.melbourne.sgi.com"
pmcd.pmlogger.port
     inst [10728 or "10728"] value 4331
```

```
pmcd.pmlogger.archive
     inst [10728 or "10728"] value "/usr/var/adm/pcplog/gonzo/foo"
```

All of the information describing the creator of the archive is revealed and, in particular, the instance identifier for the `pmcd` metrics (`10728` in the example above) is the PID of the `pmlogger` instance, which may be used to control the process via `pmlc`.

7.5.5 Illegal Label Record

Symptom: PCP tools report:

```
Illegal label record at start of PCP archive log file.
```

Cause: Either you are attempting to read a Version 2 archive with a PCP 1.*x* tool, or the archive log has become corrupted.

Resolution: By default, `pmlogger` in PCP release 2.0 and later generates Version 2 archives that PCP 1.0 to 1.3 tools cannot interpret. If you must use older tools, pass the `-V1` option to `pmlogger`, forcing it to generate Version 1 archives.

If the PCP tools are from PCP 2.0 or later, then the archive log may have been corrupted, which can be verified using `pmlogcheck`. Refer to the `pmlogcheck`(1) man page.

7.5.6 Empty Archive Log Files or `pmlogger` Exits Immediately

Symptom: Archive log files are zero size, requested metrics are not being logged, or `pmlogger` exits immediately with no error messages.

Cause: Either `pmlogger` encountered errors in the configuration file or has not flushed its output buffers yet or some (or all) metrics specified in the `pmlogger` configuration file have had their state changed to advisory `off` or mandatory `off` via `pmlc`. It is also possible that the logging interval specified in the `pmlogger` configuration file for some or all of the metrics is longer than

the period of time you have been waiting since `pmlogger` started.

Resolution: If `pmlogger` exits immediately with no error messages, check the `pmlogger.log` file in the directory `pmlogger` was started in for any error messages. If `pmlogger` has not yet flushed its buffers, enter the following command:

```
killall -SIGUSR1 pmlogger
```

Otherwise, use the `status` command for `pmlc` to interrogate the internal `pmlogger` state of specific metrics.

Performance Co-Pilot Deployment Strategies [8]

Performance Co-Pilot is a coordinated suite of tools and utilities allowing you to monitor performance and make automated judgments and initiate actions based on those judgments. PCP is designed to be fully configurable for custom implementation and deployed to meet specific needs in a variety of operational environments.

Because each enterprise and site is different and PCP represents a new way of visualizing performance information, some discussion of deployment strategies is useful.

The most common use of performance monitoring utilities is a scenario where the PCP tools are executed on a workstation (the PCP monitoring system), while the interesting performance data is collected on remote systems (PCP collector systems) by a number of processes, specifically the Performance Metrics Collection Daemon (PMCD) and the associated Performance Metric Domain Agents (PMDAs). These processes can execute on both the monitoring system and one or more collector systems, or only on collector systems. However, collector systems are the real object of performance investigations.

The material in this chapter covers the following areas:

- Section 8.1, page 178, presents the spectrum of deployment architectures at the highest level.
- Section 8.2, page 180, describes alternative deployments for PMCD and the PMDAs.
- Section 8.3, page 183, covers alternative deployments for the `pmlogger` tool.
- Section 8.4, page 185, presents the options that are available for deploying the `pmie` tool.

The options shown in this chapter are merely suggestions. They are not comprehensive, and are intended to demonstrate some possible ways of deploying the PCP tools for specific network topologies and purposes. You are encouraged to use them as the basis for planning your own deployment, consistent with your needs.

8.1 Basic Deployment

In the simplest PCP deployment, one system is configured as both a collector and a monitor, as shown in Figure 35. Because the PCP monitor tools make extensive use of visualization, this suggests the single system would be configured with a graphics head.

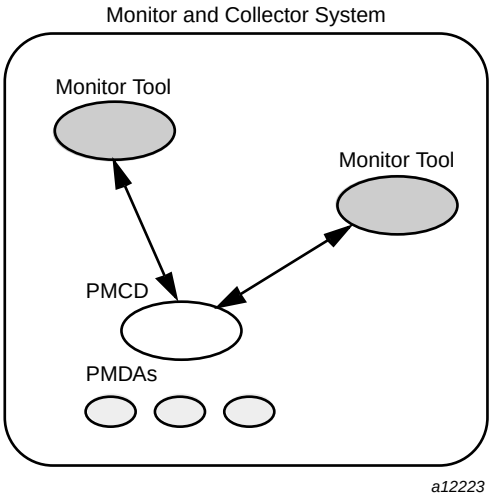

Figure 35. PCP Deployment for a Single System

However, most PCP deployments involve at least two systems. For example, the setup shown in Figure 36 would be representative of many common scenarios.

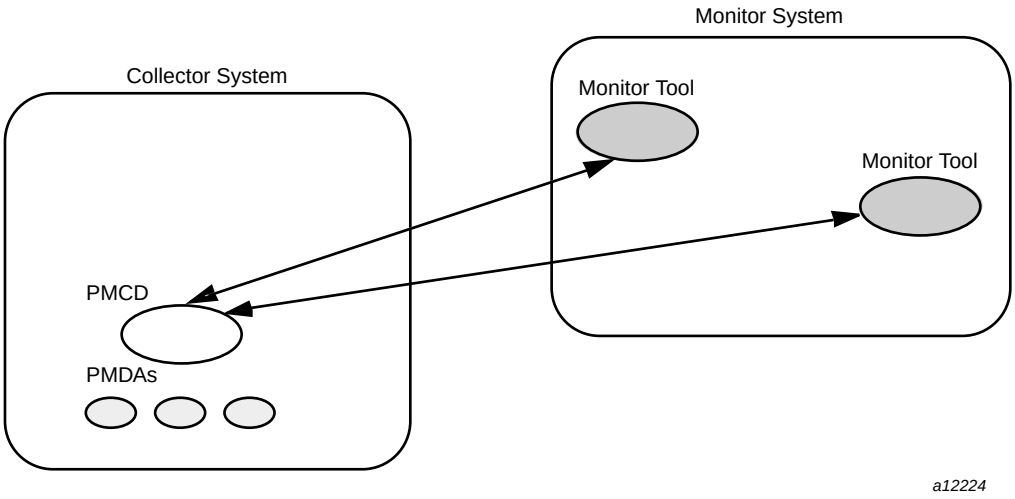

Figure 36. Basic PCP Deployment for Two Systems

But the most common site configuration would include a mixture of systems configured as PCP collectors, as PCP monitors, and as both PCP monitors and collectors, as shown in Figure 37.

With one or more PCP collector systems and one or more PCP monitor systems, there are a number of decisions that need to be made regarding the deployment of PCP services across multiple hosts. For example, in Figure 37 there are several ways in which both the inference engine (`pmie`) and the PCP archive logger (`pmlogger`) could be deployed. These options are discussed in the following sections of this chapter.

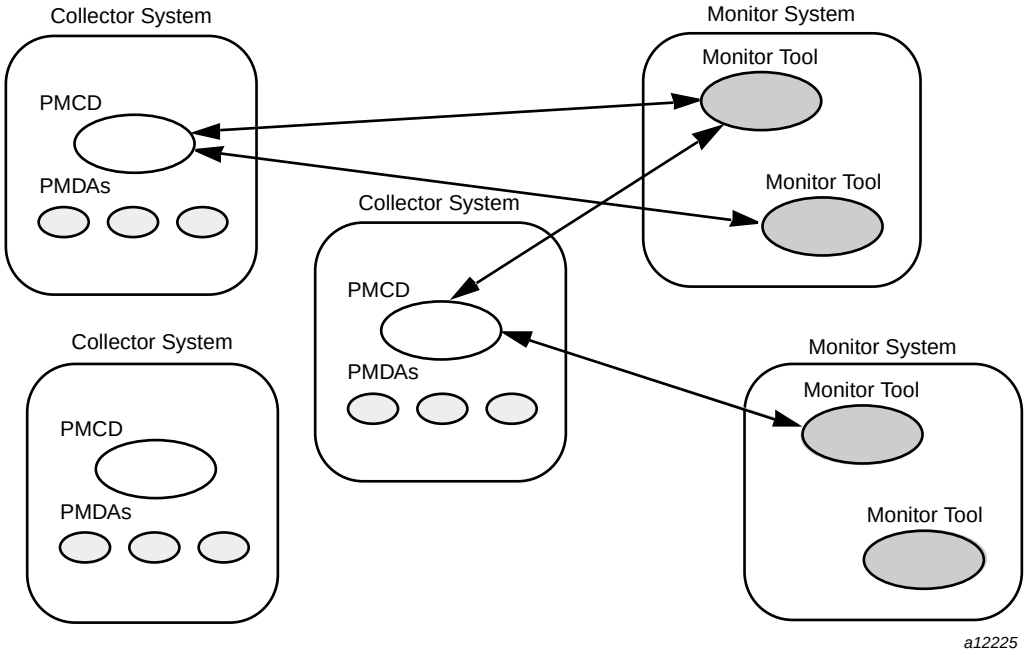

Figure 37. General PCP Deployment for Multiple Systems

8.2 PCP Collector Deployment

Each PCP collector system must have an active `pmcd` and, typically, a number of PMDAs installed.

8.2.1 Principal Server Deployment

The first hosts selected as PCP collector systems are likely to provide some class of service deemed to be critical to the information processing activities of the enterprise. These hosts include the following:

- A server running a DBMS
- A World Wide Web server for an Internet or Intranet
- An NFS file server
- A video server
- A supercomputing server

- An infrastructure service provider, for example, print, Usenet news, DNS, gateway, firewall, packet router, or mail services
- A system running a mission-critical application

Your objective may be to improve quality of service on a system functioning as a server for many clients. You wish to identify and repair critical performance bottlenecks and deficiencies in order to maintain maximum performance for clients of the server.

For some of these services, the PCP base product or the PCP add-on products provide the necessary collector components. Others would require customized PMDA development, as described in the companion *Performance Co-Pilot Programmer's Guide*.

8.2.2 Quality of Service Measurement

Applications and services with a client-server architecture need to monitor performance at both the server side and the client side.

The arrangement in Figure 38 illustrates one way of measuring quality of service for client-server applications.

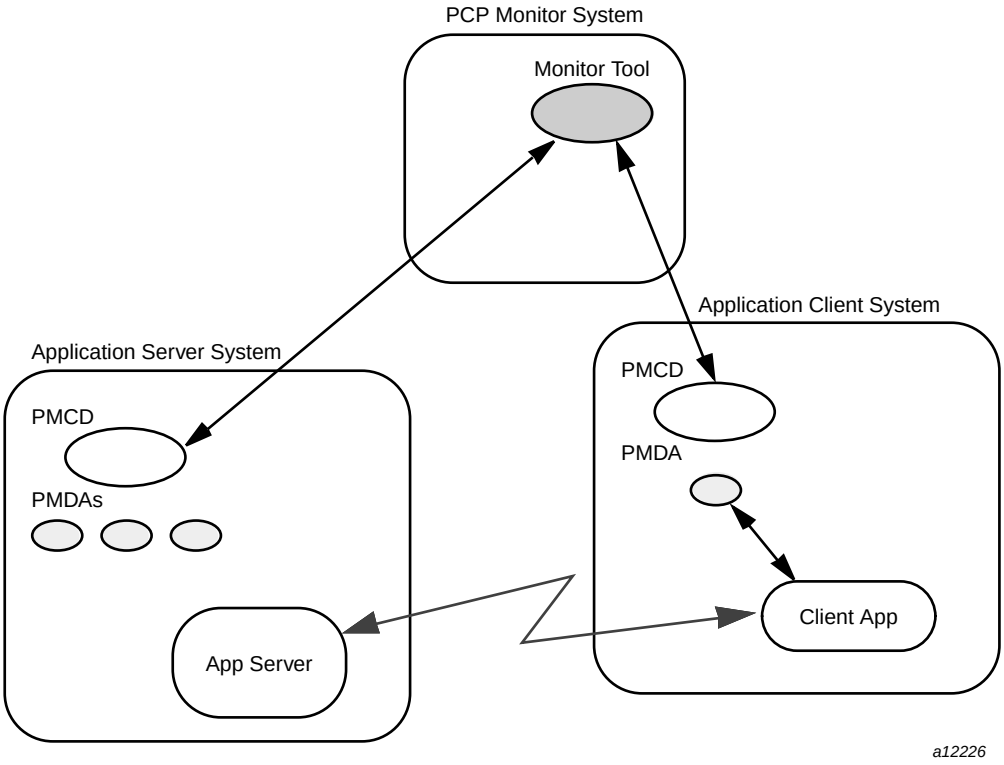

Figure 38. PCP Deployment to Measure Client-Server Quality of Service

The configuration of the PCP collector components on the Application Server System is standard. The new facility is the deployment of some PCP collector components on the Application Client System; this uses a customized PMDA and a generalization of the ICMP "ping" tool as follows:

- The `Client App` is specially developed to periodically make typical requests of the `App Server`, and to measure the response time for these requests (this is an application-specific "ping").
- The PMDA on the Application Client System captures the response time measurements from the `Client App` and exports these into the PCP framework.

At the PCP monitor system, the performance of the system running the `App Server` and the end-user quality of service measurements from the system where the `Client App` is running can be monitored concurrently.

PCP add-on products implement a number of examples of this architecture, including the `shping` PMDA for IP-based services, the `webping` PMDA for Web servers, and the `oraping` PMDA for an Oracle DBMS.

For each of these PMDAs, the full source code is distributed with the associated PCP product to encourage adaptation of the agents to the local application environment.

It is possible to exploit this arrangement even further, with these methods:

- Creating new instances of the `Client App` and PMDA to measure service quality for your own mission-critical services.
- Deploying the `Client App` and associated PCP collector components in a number of strategic hosts allows the quality of service over the enterprise's network to be monitored. For example, service can be monitored on the Application Server System, on the same LAN segment as the Application Server System, on the other side of a firewall system, or out in the WAN.

8.3 PCP Archive Logger Deployment

PCP archive logs are created by the `pmlogger` utility, as discussed in Chapter 7. They provide a critical capability to perform retrospective performance analysis, for example, to detect performance regressions, for problem analysis, or to support capacity planning. The following sections discuss the options and trade-offs for `pmlogger` deployment.

8.3.1 Deployment Options

The issue is relatively simple and reduces to "On which host(s) should `pmlogger` be running?" The options are these:

- Run `pmlogger` on each PCP collector system to capture local performance data.
- Run `pmlogger` on some of the PCP monitor systems to capture performance data from remote PCP collector systems.
- As an extension of the previous option, designate one system to act as the PCP archive site to run all `pmlogger` instances. This arrangement is shown in Figure 39.

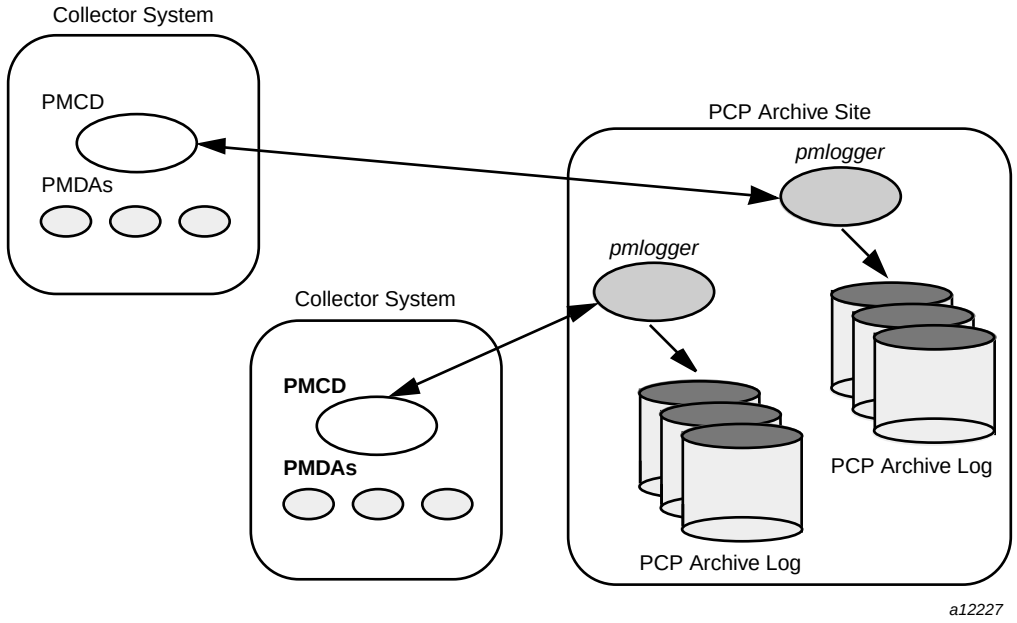

Figure 39. Designated PCP Archive Site

8.3.2 Resource Demands for the Deployment Options

The `pmlogger` process is very lightweight in terms of computational demand; so most of the (small) CPU cost associated with extracting performance metrics at the PCP collector system involves PMCD and the PMDAs, which are independent of the host on which `pmlogger` is running.

A local `pmlogger` consumes disk bandwidth and disk space on the PCP collector system. A remote `pmlogger` consumes disk space on the site where it is running and network bandwidth between that host and the PCP collector host.

The archive logs typically grow at the rate of between 500 kilobytes (KB) and 10 megabytes (MB) per day, depending on how many performance metrics are logged and the choice of sampling frequencies. There are some advantages in minimizing the number of hosts over which the disk resources for PCP archive logs must be allocated; however, the aggregate requirement is independent of where the `pmlogger` instances are running.

8.3.3 Operational Management

There is an initial administrative cost associated with configuring each `pmlogger` instance, and an ongoing administrative investment to monitor these configurations, perform regular housekeeping (such as rotation, compression, and culling of PCP archive log files), and execute periodic tasks to process the archives (such as nightly performance regression checking with `pmie`, or using `pmchart` to publish recent activity charts on the Web).

Many of these tasks are handled by the supplied `pmlogger` administrative tools and scripts, as described in Section 7.2.3. However, the necessity and importance of these tasks favor a centralized `pmlogger` deployment, as shown in Figure 39.

Note: The `pmlogger` utility is not subject to any PCP license restrictions, and may be installed and used on any host.

8.3.4 Exporting PCP Archive Logs

Collecting PCP archive logs is of little value unless the logs are processed as part of the ongoing performance monitoring and management functions. This processing typically involves the use of the tools on a PCP monitor system, and hence the archive logs may need to be read on a host different from the one they were created on.

NFS mounting is obviously an option, but the PCP tools support random access and both forward and backward temporal motion within an archive log. If an archive is to be subjected to intensive and interactive processing, it may be more efficient to copy the files of the archive log to the PCP monitor system first.

Note: Each PCP archive log consists of at least three separate files (see Section 7.2.3 for details). You must have concurrent access to all of these files before a PCP tool is able to process an archive log correctly.

8.4 PCP Inference Engine Deployment

The `pmie` utility supports automated reasoning about system performance, as discussed in Chapter 6, and plays a key role in monitoring system performance for both real-time and retrospective analysis, with the performance data being retrieved respectively from a PCP collector system and a PCP archive log.

The following sections discuss the options and trade-offs for `pmie` deployment.

8.4.1 Deployment Options

The issue is relatively simple and reduces to "On which host(s) should `pmie` be running?" You must consider both real-time and retrospective uses, and the options are as follows:

- For real-time analysis, run `pmie` on each PCP collector system to monitor local system performance.
- For real-time analysis, run `pmie` on some of the PCP monitor systems to monitor the performance of remote PCP collector systems.
- For retrospective analysis, run `pmie` on the systems where the PCP archive logs reside. The problem then reduces to `pmlogger` deployment as discussed in Section 8.3.
- As an example of the "distributed management with centralized control" philosophy, designate some system to act as the PCP Management Site to run all `pmlogger` and `pmie` instances. This arrangement is shown in Figure 40.

One `pmie` instance is capable of monitoring multiple PCP collector systems; for example, to evaluate some universal rules that apply to all hosts. At the same time a single PCP collector system may be monitored by multiple `pmie` instances; for example, for site-specific and universal rule evaluation, or to support both tactical performance management (operations) and strategic performance management (capacity planning). Both situations are depicted in Figure 40.

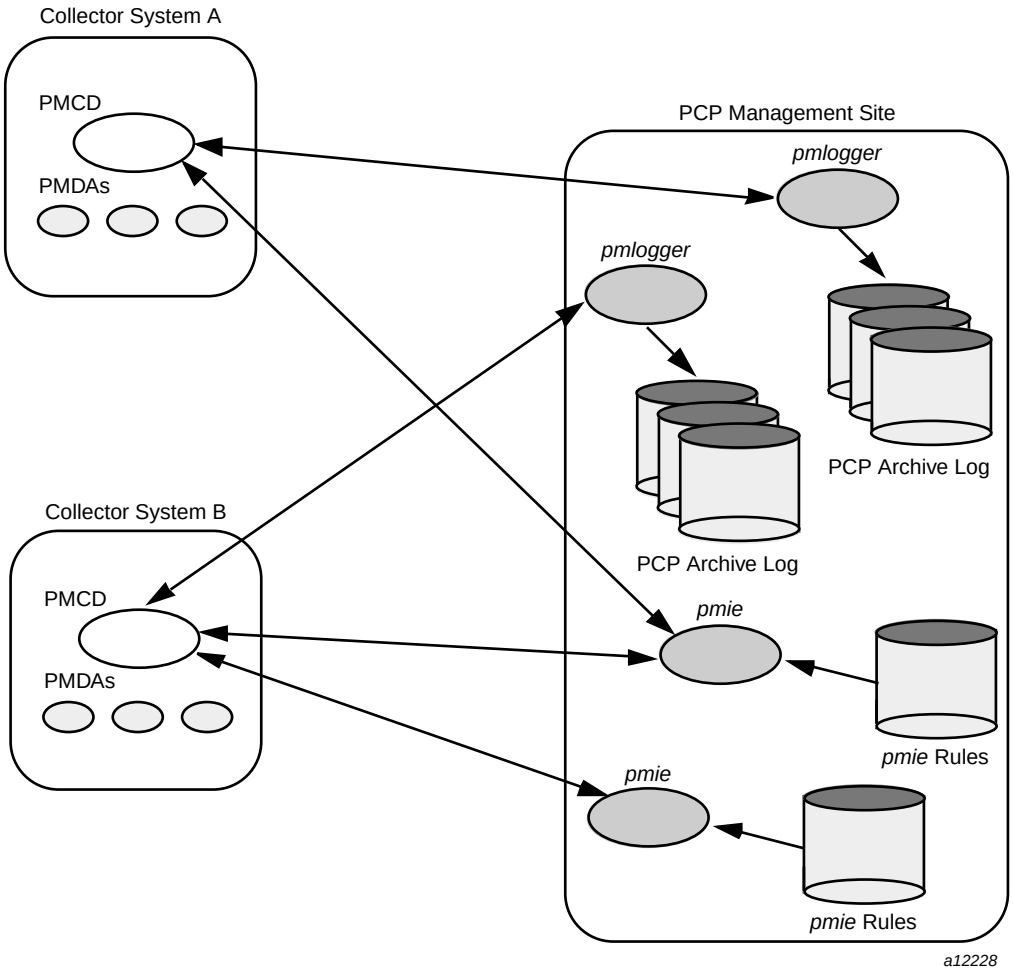

Figure 40. PCP Management Site Deployment

8.4.2 Resource Demands for the Deployment Options

Depending on the complexity of the rule sets, the number of hosts being monitored, and the evaluation frequency, `pmie` may consume CPU cycles significantly above the resources required to simply fetch the values of the performance metrics. If this becomes significant, then real-time deployment of `pmie` away from the PCP collector systems should be considered in order to avoid the "you're part of the problem, not the solution" scenario in terms of CPU utilization on a heavily loaded server.

8.4.3 Operational Management

An initial administrative cost is associated with configuring each `pmie` instance, particularly in the development of the rule sets that accurately capture and classify "good" versus "bad" performance in your environment. These rule sets almost always involve some site-specific knowledge, particularly in respect to the "normal" levels of activity and resource consumption. The `pmieconf` tool (see Section 6.7, page 143) may be used to help develop localized rules based upon parameterized templates covering many common performance scenarios. In complex environments, customizing these rules may occur over an extended period and require considerable performance analysis insight.

One of the functions of `pmie` provides for continual detection of adverse performance and the automatic generation of alarms (visible, audible, e-mail, pager, and so on). Uncontrolled deployment of this alarm initiating capability throughout the enterprise may cause havoc.

These considerations favor a centralized `pmie` deployment at a small number of PCP monitor sites, or in a PCP Management Site as shown in Figure 40.

However, it is most likely that knowledgeable users with specific needs may find a local deployment of `pmie` most useful to track some particular class of service difficulty or resource utilization. In these cases, the alarm propagation is unlikely to be required or is confined to the system on which `pmie` is running.

Configuration and management of a number of `pmie` instances is made much easier with the scripts and control files described in Section 6.9, page 150.

Customizing and Extending PCP Services [9]

Performance Co-Pilot (PCP) has been developed to be fully extensible. The following sections summarize the various facilities provided to allow you to extend and customize PCP for your site:

- Section 9.1, page 189, describes the general process of installing and removing a PMDA at both a PCP collector and/or a PCP monitor host. It also describes the procedure for customizing the summary PMDA to export derived metrics formed by aggregation of base PCP metrics from one or more collector hosts.
- Section 9.2, page 193, describes the various options available for customizing and extending the basic PCP tools.
- Section 9.3, page 198, covers the concepts and tools provided for updating the PMNS (Performance Metrics Name Space).
- Section 9.4, page 201, details where to find further information to assist in the development of new PMDAs to extend the range of performance metrics available through the PCP infrastructure.
- Section 9.5, page 201, outlines how new tools may be developed to process performance data from the PCP infrastructure.

9.1 PMDA Customization

The generic procedures for installing and activating the optional PMDAs have been described in Section 2.5, page 34. In some cases, these procedures prompt the user for information based upon the local system or network configuration, application deployment, or processing profile to customize the PMDA and hence the performance metrics it exports.

The summary PMDA is a special case that warrants further discussion.

9.1.1 Customizing the Summary PMDA

The summary PMDA exports performance metrics derived from performance metrics made available by other PMDAs. It is described completely in the `pmdasummary`(1) man page.

The summary PMDA consists of two processes:

`pmie` process	Periodically samples the base metrics and compute values for the derived metrics. This dedicated instance of the PCP `pmie` inference engine is launched with special command line arguments by the main process. See Section 6.1, page 117, for a complete discussion of the `pmie` feature set.
`main` process	Reads and buffers the values computed by the `pmie` process and makes them available to the Performance Metrics Collection Daemon (PMCD).

All of the metrics exported by the summary PMDA have a singular instance and the values are instantaneous; the exported value is the correct value as of the last time the corresponding expression was evaluated by the `pmie` process.

The summary PMDA resides in the `/usr/pcp/pmdas/summary` directory and may be installed with a default configuration by following the steps described in Section 2.5.1, page 34.

Alternatively, you may customize the summary PMDA to export your own derived performance metrics by following the steps in Procedure 6:

Procedure 6: Customizing the Summary PMDA

1. Check that the symbolic constant SYSSUMMARY is defined in the `/var/pcp/pmns/stdpmid` file. If it is not, perform the postinstall update of this file, as superuser:

   ```
   # cd /var/pcp/pmns
           # ./Make.stdpmid
   ```

2. Choose Performance Metric Name Space (PMNS) names for the new metrics. These must begin with `summary` and follow the rules described in the `pmns`(4) man page. For example, you might use `summary.fs.cache_write` and `summary.fs.cache_hit`.

3. Edit the `pmns` file in the `/usr/pcp/pmdas/summary` directory to add the new metric names in the format described in the `pmns`(4) man page. You must choose a unique Performance Metric Identifier (PMID) for each metric. In the `pmns` file, these appear as *SYSSUMMARY:0:x*. The value of *x* is arbitrary in the range 0 to 1023 and unique in this file. Refer to Section 9.3, page 198, for a further explanation of the rules governing PMNS updates.

   For example:

```
summary {
          cpu
          disk
          netif
          fs              /*new*/
}
summary.fs {
          cache_write     SYSSUMMARY:0:10
          cache_hit       SYSSUMMARY:0:11
}
```

4. Use the local test PMNS `root` and validate that the PMNS changes are correct.

   For example, enter this command:

   `pminfo -n root -m summary.fs`

   You see output similar to the following:

   ```
   summary.fs.cache_write PMID: 27.0.10
   summary.fs.cache_hit PMID: 27.0.11
   ```

5. Edit the `/usr/pcp/pmdas/summary/expr.pmie` file to add new `pmie` expressions. If the name to the left of the assignment operator (=) is one of the PMNS names, then the `pmie` expression to the right will be evaluated and returned by the summary PMDA. The expression must return a numeric value. Additional description of the `pmie` expression syntax may be found in Section 6.3, page 124.

   For example, consider this expression:

   ```
   // filesystem buffer cache hit percentages
   prefix = "kernel.all.io";          // macro, not exported
   summary.fs.cache_write =
                   100 - 100 * $prefix.bwrite / $prefix.lwrite;
   summary.fs.cache_hit =
                   100 - 100 * $prefix.bread / $prefix.lread;
   ```

6. Run `pmie` in debug mode to verify that the expressions are being evaluated correctly, and the values make sense.

   For example, enter this command:

   `pmie -t 2sec -v expr.pmie`

   You see output similar to the following:

```
summary.fs.cache_write:      ?
summary.fs.cache_hit:      ?
summary.fs.cache_write:  45.83
summary.fs.cache_hit:   83.2
summary.fs.cache_write:  39.22
summary.fs.cache_hit:  84.51
```

7. Install the new PMDA.

   From the `/usr/pcp/pmdas/summary` directory, use this command:

   `./Install`

   You see the following output:

```
You need to choose an appropriate configuration for installation of
the ''summary'' Performance Metrics Domain Agent (PMDA).

collector collect performance statistics on this system
monitor   allow this system to monitor local and/or remote systems
both      collector and monitor configuration for this system

Please enter c(ollector) or m(onitor) or b(oth) [b] both
Interval between summary expression evaluation (seconds)? [10] 10
Updating the Performance Metrics Name Space...
Installing pmchart view(s) ...
Terminate PMDA if already installed ...
Installing files ..
Updating the PMCD control file, and notifying PMCD ...
Wait 15 seconds for the agent to initialize ...
Check summary metrics have appeared ... 8 metrics and 8 values
```

8. Check the metrics.

   For example, enter this command:

   `pmval -t 5sec -s 8 summary.fs.cache_write`

   You see a response similar to the following:

```
metric:    summary.fs.cache_write
host:      localhost
semantics: instantaneous value
units:     none
samples:   8
interval:  5.00 sec
```

```
63.60132158590308
62.71878646441073
62.71878646441073
58.73968492123031
58.73968492123031
65.33822758259046
65.33822758259046
72.6099706744868
```

Note that the values are being sampled here by `pmval` every 5 seconds, but `pmie` is passing only new values to the summary PMDA every 10 seconds. Both rates could be changed to suit the dynamics of your new metrics.

9. You may now create `pmchart` views, `pmview` scenes, and `pmlogger` configurations to monitor and archive your new performance metrics.

9.2 PCP Tool Customization

Performance Co-Pilot (PCP) has been designed and implemented with a philosophy that embraces the notion of toolkits and encourages extensibility.

In most cases, the PCP tools provide orthogonal services, based on external configuration files. It is the creation of new and modified configuration files that enables PCP users to customize tools quickly and meet the needs of the local environment, in many cases allowing personal preferences to be established for individual users on the same PCP monitor system.

The material in this section is intended to act as a checklist of pointers to detailed documentation found elsewhere in this guide, in the man pages, and in the files that are made available as part of the PCP installation.

9.2.1 Stripchart Customization

The PCP tool `pmchart` produces stripchart displays of performance metrics. Refer to Section 4.1, page 62, for an extensive description of the capabilities of `pmchart`.

Customization is centered on PCP views that may be created interactively and saved via the `Save View` option in the `File` menu.

When `pmchart` is loading a view, the following directories are searched:

`.`	The current directory.
`$HOME/.pcp`	Views for each user.

`/var/pcp/config/pmchart` — The system-wide catalog of views. Any view installed here becomes visible to every `pmchart` user.

The X11 application resources for `pmchart` are in `/usr/lib/X11/app-defaults/PmChart`, and these may be edited to customize the appearance of the display. The default update interval and other attributes are described in the `pmchart`(1) man page.

9.2.2 Archive Logging Customization

The PCP archive logger is presented in Chapter 7, page 155, and documented in the `pmlogger`(1) man page.

The following global files and directories influence the behavior of `pmlogger`:

`/etc/config/pmlogger`

Enable/disable state for the primary logger facility using this command:

```
chkconfig pmlogger on
```

`/etc/config/pmlogger.options`

Command line options passed to the primary logger if it is launched from `/etc/init.d/pcp`.

`/var/pcp/config/pmlogger/config.default`

The default `pmlogger` configuration file that is used for the primary logger when this facility is enabled.

`/var/pcp/config/pmlogger/config.view.*`

Every `pmchart` view also provides a `pmlogger` configuration file that includes each of the performance metrics used in the view, for example, `/var/pcp/config/pmlogger/config.LoadAvg` for the LoadAvg view.

`/var/pcp/config/pmlogger/config.*`

Every PCP tool with a fixed group of performance metrics contributes a `pmlogger` configuration file that includes each of

the performance metrics used in the tool, for example, `/var/pcp/config/pmlogger/config.dkvis` for `dkvis`.

`/var/pcp/config/pmlogger/control`

Defines which PCP collector hosts require `pmlogger` to be launched on the local host, where the configuration file comes from, where the archive log files should be created, and `pmlogger` startup options.

`/var/pcp/config/pmlogger/crontab`

Prototype `crontab` entries that may be merged with the `crontab` entries for `root` to schedule the periodic execution of the archive log management scripts, for example, `pmlogger.daily`.

`/var/adm/pcplog/somehost`

The default behavior of the archive log management scripts create archive log files for the host *somehost* in this directory.

`/var/adm/pcplog/somehost/Latest`

A PCP archive folio for the most recent archive for the host *somehost*. This folio is created and maintained by the `cron`-driven periodic archive log management scripts, for example, `pmlogger.check`. Archive folios may be processed with the `pmafm` tool.

9.2.3 Inference Engine Customization

The PCP inference engine is presented in Chapter 6, page 117, and documented in the `pmie`(1) man page.

The following global files and directories influence the behavior of `pmie`:

`/etc/config/pmie`

Controls the pmie daemon facility. Enable using this command:

```
chkconfig pmie on
```

`/var/pcp/demos/pmie/*`

Example `pmie` rules that may be used as a basis for developing local rules.

`/var/pcp/config/pmie/config.default`

The default `pmie` configuration file that is used when the `pmie` daemon facility is enabled.

`/var/pcp/config/pmieconf/*/*`

Each `pmieconf` rule definition can be found below one of these subdirectories.

`/var/pcp/config/pmie/control`

Defines which PCP collector hosts require a daemon `pmie` to be launched on the local host, where the configuration file comes from, where the `pmie` log file should be created, and `pmie` startup options.

`/var/pcp/config/pmlogger/crontab`

Prototype `crontab` entries that may be merged with the `crontab` entries for root to schedule the periodic execution of the `pmie_check` script, for verifying that `pmie` instances are running.

`/var/adm/pmielog/`

The default behavior of the `/etc/init.d/pmie` startup scripts create `pmie` log files for the host in this directory.

The `pmcd` PMDA exports information about executing `pmie` instances and their progress in terms of rule evaluations and action execution rates.

`pmie_check`	This command is similar to the `pmlogger` support script, `pmlogger_check`.
`/etc/init.d/pmie`	This control file supports the starting and stopping of multiple `pmie` instances that are monitoring one or more hosts.
`/var/tmp/pmie`	The statistics that `pmie` gathers are maintained in binary data structure files. These files are in the `/var/tmp/pmie` directory.

`pmcd.pmie` metrics

If `pmie` is running on a system with a PCP collector deployment, the `pmcd` PMDA exports these metrics via the `pmcd.pmie` group of metrics.

9.2.4 Snapshot Customization

The PCP snapshot production facility is presented in Section 7.3.4, page 166, and documented in the `pmsnap`(1) man page.

The following global files and directories influence the behavior of `pmsnap`:

`/var/pcp/config/pmsnap/control`

Defines how to produce a snapshot, including the output filename, the PCP archive folio name to be used as input, the `pmchart` configuration file, and command line arguments to `pmchart`

`/var/pcp/config/pmsnap/Summary`

A `pmchart` configuration file to produce a sample summary snapshot in conjunction with `pmsnap`

`/var/pcp/config/pmlogger/config.Summary`

A `pmlogger` configuration file that can produce an archive containing performance metrics required by the sample summary snapshot

`/var/pcp/config/pmlogger/crontab`

Prototype `crontab` entries that may be merged with the `crontab` entries for `root` schedule the periodic execution of the archive log management scripts, for example, `pmsnap`

`/var/pcp/config/pmsnap/Summary.html`

An example HTML page suitable for publishing images from the `pmsnap` examples via a Web server

9.2.5 Icon Control Panel Customization

The gadget specification language of `pmgadgets` supports the creation of arbitrary gadget layouts and bindings to hosts and performance metrics. See Section 4.2, page 81, and the `pmgadgets`(1) man page.

9.2.6 3D Visualization Customization

The 3D scene specification language of `pmview` supports the creation of block layouts and bindings to hosts and performance metrics. See Chapter 5 and the `pmview`(1) man page.

9.3 PMNS Management

This section describes the syntax, semantics, and processing framework for the external specification of a Performance Metrics Name Space (PMNS) as it might be loaded by the PMAPI routine `pmLoadNameSpace`; see the `pmLoadNameSpace`(3) man page.

The PMNS specification is a simple ASCII source file that can be edited easily. For reasons of efficiency, a binary format is also supported; the utility `pmnscomp` translates the ASCII source format into binary format; see the `pmnscomp`(1) man page.

9.3.1 PMNS Processing Framework

The PMNS specification is initially passed through `cpp`. This means the following facilities may be used in the specification:

- C-style comments
- #include directives
- #define directives and macro substitution
- Conditional processing with #if, #endif, and so on

When `cpp` is executed, the standard include directories are the current directory and `/var/pcp/pmns`, where some standard macros and default specifications may be found.

9.3.2 PMNS Syntax

Every PMNS is tree structured. The paths to the leaf nodes are the performance metric names. The general syntax for a non-leaf node in PMNS is as follows:

```
pathname {
name [pmid]
...
}
```

Here `pathname` is the full pathname from the root of the PMNS to this non-leaf node, with each component in the path separated by a period. The root node for the PMNS has the special name `root`, but the prefix string `root.` must be omitted from all other `pathnames`.

For example, refer to the PMNS shown in Figure 41. The correct pathname for the rightmost non-leaf node is `cpu.util`, not `root.cpu.util`.

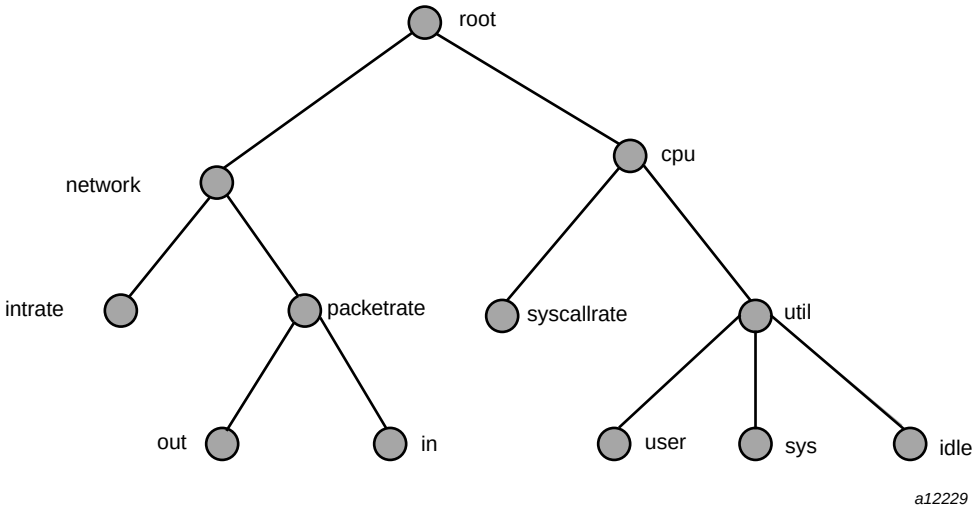

Figure 41. Small Performance Metrics Name Space (PMNS)

Each component in the pathname must begin with an alphabetic character and be followed by zero or more alphanumeric characters or the underscore (_) character. For alphabetic characters in a component, uppercase and lowercase are significant.

Non-leaf nodes in the PMNS may be defined in any order desired. The descendent nodes are defined by the set of `names`, relative to the pathname of their parent non-leaf node. For descendent nodes, leaf nodes have a `pmid` specification, but non-leaf nodes do not.

The syntax for the `pmid` specification was chosen to help manage the allocation of Performance Metric IDs (PMIDs) across disjoint and autonomous domains of administration and implementation. Each `pmid` consists of three integers separated by colons, for example, `14:27:11`. This is intended to mirror the implementation hierarchy of performance metrics. The first integer identifies

the domain in which the performance metric lies. Within a domain, related metrics are often grouped into clusters. The second integer identifies the cluster, and the third integer, the metric within the cluster.

The PMNS specification for Figure 41 is shown in Example 7, page 200:

Example 7: PMNS Specification

```
/*
    * PMNS Specification
    */
#include <stdpmid>
root {
       network
       cpu
}
#define NETWORK 26
network {
       intrate    ':NETWORK:1
       packetrate
}
network.packetrate {
       in     IRIX:NETWORK:35
       out    IRIX:NETWORK:36
}
#define CPU 10
cpu {
       syscallrate    IRIX:CPU:10
       util
}
#define USER 20
#define KERNEL 21
#define IDLE 22
cpu.util {
       user    IRIX:CPU:USER
       sys     IRIX:CPU:KERNEL
       idle    IRIX:CPU:IDLE
}
```

For complete documentation of the PMNS and associated utilities, see the pmns(4), pmnscomp(1), pmnsadd(1), pmnsdel(1), and pmnsmerge(1) man pages.

9.4 PMDA Development

Performance Co-Pilot (PCP) is designed to be extensible at the collector site.

Application developers are encouraged to create new PMDAs to export performance metrics from the applications and service layers that are particularly relevant to a specific site, application suite, or processing environment.

These PMDAs use the routines of the `libpcp_pmda` library, which is discussed in detail by the *Performance Co-Pilot Programmer's Guide*.

Source code for several PMDAs (simple, trivial, and txmon) is provided in the `pcp.sw.demo` subsystem. When it is installed, all of the relevant files reside in directories (one per PMDA) below the `/var/pcp/pmdas` directory.

9.5 PCP Tool Development

Performance Co-Pilot (PCP) is designed to be extensible at the monitor site.

Application developers are encouraged to create new PCP client applications to monitor or display performance metrics in a manner that is particularly relevant to a specific site, application suite, or processing environment.

Client applications use the routines of the PMAPI (performance metrics application programming interface) described in the *Performance Co-Pilot Programmer's Guide*.

Source code for a sample PMAPI client (`pmclient`) is provided in the `pcp.sw.demo` subsystem, and when installed all of the relevant files reside in `/var/pcp/demos/pmclient`.

Acronyms [A]

This appendix provides a list of the acronyms used in the Performance Co-Pilot (PCP) documentation, help cards, man pages, and user interface.

Table 4. Performance Co-Pilot Acronyms and Their Meanings

Acronym	Meaning
API	Application Programming Interface
DBMS	Database Management System
DNS	Domain Name Service
DSO	Dynamic Shared Object
IP	Internet Protocol
PCP	Performance Co-Pilot
PDU	Protocol Data Unit
PMAPI	Performance Metrics Application Programming Interface
PMCD	Performance Metrics Collection Daemon
PMCS	Performance Metrics Collection Subsystem
PMD	Performance Metrics Domain
PMDA	Performance Metrics Domain Agent
PMID	Performance Metric Identifier
PMNS	Performance Metrics Name Space
TCP/IP	Transmission Control Protocol/Internet Protocol

Index

D

E

M

N

O

P