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

Document Number: 007-2614-003

CONTRIBUTORS

Engineering and written contributions by David Chatterton, Mark Goodwin, Ken McDonell, Ania Milewska, Nathan Scott, Timothy Shimmin, and Bill Tuthill.
Production by Linda Rae Sande.
St. Peter's Basilica image courtesy of ENEL SpA and InfoByte SpA. Disk Thrower image courtesy of Xavier Berenguer, Animatica.



Performance Co-Pilot™ User's and Administrator's Guide
Document Number: 007-2614-003

# Contents

# List of Figures

# List of Tables

# About This Guide

This guide describes the Performance Co-Pilot (PCP) software package of advanced performance tools for the Silicon Graphics family of graphical workstations and servers. 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, "Introduction to Performance Co-Pilot," provides a brief overview of the software components and conceptual foundations of the PCP product.
- Chapter 2, "Installing and Configuring Performance Co-Pilot," describes the basic installation and configuration steps necessary to get PCP running on your systems.
- Chapter 3, "Common Conventions and Arguments," 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, "Monitoring System Performance," describes the basic interactive performance monitoring tools available in PCP, including *pmchart*, *pmgadgets*, *pmkstat*, *pmval*, *pmdumptext*, *pmem*, *pminfo,* and *pmstore*.
- Chapter 5, "System Performance Visualization Tools," discusses the various 3D visualization tools that are provided to enable high-level monitoring, management, and diagnosis for performance problems.
- Chapter 6, "Performance Metrics Inference Engine," 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, “Archive Logging,” covers the PCP services and utilities that support archive logging for capturing accurate historical performance records.
- Chapter 8, “Performance Co-Pilot Deployment Strategies,” presents the various options for deploying PCP functionality across systems spanning the enterprise.
- Chapter 9, “Customizing and Extending PCP Services,” 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, “Acronyms,” provides a comprehensive list of the acronyms used in this guide, in the reference 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 and the *Personal System Administration Guide* as hardcopy 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.

### Reference Pages

The IRIX reference pages (also called “man” or manual pages) provide concise reference information on the use of IRIX commands, subroutines, and system resources. There is usually a reference page for each PCP command or subroutine. To see a list of all the PCP reference pages, enter the following command:

```
man -k performance
```

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

```
man pcp
```

The reference pages are divided into seven sections, as shown in Table i. When referring to reference pages, this guide follows a standard UNIX convention: the section number in parentheses follows the item. For example, PMDA(3) refers to the reference page in section 3 for the *pmda* command.

**Table i** Outline of Reference Page Organization

| Type of Reference Page | Section Number |
|---|---|
| General commands | (1) |
| System calls and error numbers | (2) |
| Library subroutines | (3) |
| File formats | (4) |
| Miscellaneous | (5) |
| Demos and games | (6) |
| Special files | (7) |

## Release Notes

Release notes provide specific information about the current release, available online through the *relnotes* 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.

## Silicon Graphics Web Sites

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

http://www.sgi.com
: The Silicon Graphics general Web site, with search capability.

http://www.sgi.com/Products
: Contains links to Performance Co-Pilot product information.

http://techpubs.sgi.com/library
: The Silicon Graphics Technical Publications Library.

## Conventions Used in This Guide

These type conventions and symbols are used in this guide:

| | |
|---|---|
| **Bold** | Literal command-line arguments (options/flags), nonalphabetic data types, operators, and subroutines. |
| *Italics* | Backus-Naur Form entries, command monitor commands, executable names, filenames, IRIX commands, manual/book titles, new terms, onscreen button names, program variables, tools, utilities, variable command-line arguments, variable coordinates, and variables to be supplied by the user in examples, code, and syntax statements |
| `Fixed-width type` | Error messages, prompts, and onscreen text |
| **`Bold fixed-width type`** | User input, including keyboard keys (printing and nonprinting); literals supplied by the user in examples, code, and syntax statements |
| ALL CAPS | Environment variables, operator names, directives, defined constants, macros in C programs |
| “” | (Double quotation marks) Onscreen menu items and references to document section titles within text |
| () | (Parentheses) Following function names—surround function arguments or are empty if the function has no arguments; following IRIX commands—surround reference page (man page) section numbers. |
| [] | (Brackets) Surrounding optional syntax statement arguments |
| `#` | IRIX shell prompt for the superuser (*root*) |
| `%` | IRIX shell prompt for users other than the superuser |
| `>>` | Command Monitor prompt |

*Chapter 1*

# Introduction to Performance Co-Pilot

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:

- "Objectives of Performance Co-Pilot" covers the intended purposes of PCP.
- "Overview of Component Software" on page 5 describes PCP tools and agents.
- "Conceptual Foundations" on page 11 discusses and design theories behind PCP.

## Objectives of Performance Co-Pilot

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

### PCP Target Usage

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

### Empowering the PCP User

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

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

## Unification of Performance Metric Domains

At the lowest level, performance metrics are collected and managed in autonomous performance domains such as the 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 Metric Domain Agents (PMDAs) within PCP encapsulate the knowledge about, and export performance information from, autonomous performance domains.

## Uniform Naming and Access to Performance Metrics

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

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

## PCP Distributed Operation

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

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

## Dynamic Adaptation to Change

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

The distributed nature of the PCP (and the modular fashion in which performance metrics domains can be installed, upgraded, and configured on different hosts) enables PCP to adapt concurrently to changes in the monitored system(s). Variations in the available performance metrics as a consequence of configuration changes are handled automatically and become visible to all clients as soon as the reconfigured host is rebooted or the responsible agent is restarted.

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

## Logging and Retrospective Analysis

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

A universal replay mechanism, modeled on VCR controls, supports “stop, rewind, random seek, and replay 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 service, provides new and powerful ways to build performance tools and to review both current and historical performance data.

## Automated Operational Support

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

- centralized archive logging for multiple remote hosts
- archive log rotation, consolidation and culling

- Web-based publishing of charts showing snapshots of performance activity levels in the recent past
- flexible alarm monitoring for critical performance scenarios
- retrospective performance audits covering the recent past; for example, daily or weekly checks for performance regressions or quality of service problems

## PCP Extensibility

PCP permits the integration of new performance metrics into the 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.

## Additional PCP Features

The following features are included in this release of Performance Co-Pilot:

Metric Coverage
: PCP supports domains of performance metrics that include all IRIX 5.3 (and later) kernel instrumentation, process-level resource utilization, customizable summaries of performance metrics, *sendmail* queue length, 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, activity levels within the PCP infrastructure, and Cisco router statistics. 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.

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
- common DBMS products
- Silicon Graphics array platforms
- Silicon Graphics 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.

Secure Operation

A host-based security model provides optional control over execution of PCP service requests from designated remote hosts or workstations.

## Overview of Component Software

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

```
man pcp
```

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 subsections.

## Performance Monitoring and Visualization

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

| | |
|---|---|
| *cachemiss* | The *cachemiss* tool may be used to measure the cycle-time penalties associated with operations that "miss" in the primary and secondary data caches of the processor-memory hierarchy. |
| *dkvis* | The *dkvis* tool displays a three-dimensional bar chart showing activity in the disk subsystem.† |
| *mpvis* | The *mpvis* tool displays a three-dimensional bar chart of multiprocessor CPU utilization.† |
| *nfsvis* | The *nfsvis* tool displays a three-dimensional bar chart showing NFS (Network File System) client and server request activity, for systems on which the optional NFS software product has been installed.† |
| *osvis* | The *osvis* tool displays three-dimensional bar charts covering many aspects of system performance, including disk use, job load, memory, CPU activity, and network I/O.† |
| *oview* | The *oview* tool visualizes the performance of Origin systems, showing a dynamic display of Origin node topology and performance. |
| *xbowvis* | The *xbowvis* tool visualizes the Crossbow (XBow) packet and error rates on platforms that support this hardware.† |
| *pmchart* | The *pmchart* tool displays trends over time for arbitrarily selected performance metrics from one or more hosts, or from one or more performance metric domains. |
| *pmdumpmineset* | The *pmdumpmineset* tool is a wrapper for *pmdumptext* that produces data files suitable for importing into the MineSet data mining product. |
| *pmdumptext* | The *pmdumptext* command outputs the values of performance metrics collected live or from a PCP archive, as ASCII text. |
| *pmem* | The *pmem* command reports per-process memory usage statistics. Both virtual size and pro-rated physical memory usage are reported. |

† This is one of the front-end wrappers for *pmview*.

| | |
|---|---|
| *pmgadgets* | The *pmgadgets* command 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.‡ |
| *pmgevctr* | The *pmgevctr* tool uses pmgadgets to display an animated gadget that reports activity in the CPU and memory subsystems. |
| *pmgirix* | The *pmgirix* command 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* | The *pmie* tool is an inference engine to evaluate predicate-action rules over performance metrics domain, for performance alarms, automated system management tasks, dynamic tuning configuration, and so on. |
| *pminfo* | The *pminfo* command displays information about arbitrary performance metrics available from PCP, including help text with **-T**. |
| *pmkstat* | The *pmkstat* command provides a text-based display of metrics that summarize system performance at a high level, suitable for ASCII logs or enquiry over a modem. |
| *pmsocks* | The *pmsocks* command allows the execution of PCP tools through a network firewall system provided *sockd* services are supported. |
| *pmtime* | The *pmtime* command provides a graphical user interface for PCP applications requiring time control.‡ |
| *pmval* | The *pmval* command provides text-based display of the values for arbitrary instances of a selected performance metric, suitable for ASCII logs or enquiry over a modem. |
| *pmview* | The *pmview* tool is a generalized three dimensional Open Inventor application that supports dynamic displays of clusters of related performance metrics as groups of utilization blocks (or towers) on a common base plane.‡ |
| *psmon* | The *psmon* script selects a subset of the actively running processes and launches either *pmchart* or *pmlogger* to collect per-process metrics for those processes. |

‡ This command is not normally invoked directly by users.

## Collecting, Transporting, and Archiving Performance Information

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

| | |
|---|---|
| *mkaf* | The *mkaf* tool aggregates an arbitrary collection of PCP archive logs into a "folio" to be used with *pmafm*. |
| *pmafm* | The *pmafm* tool is used to interrogate, manage, and replay an archive folio as created by *mkaf*, or the periodic archive log management scripts, or the "record" mode of other PCP tools. |
| *pmcd* | 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. |
| *pmdacisco* | A Performance Metrics Domain Agent (PMDA) that extracts performance metrics from one or more Cisco routers. |
| *pmdahotproc* | A PMDA that exports performance metrics from an instance domain of processes restricted to an interesting or "hot" set. |
| *pmdamailq* | A PMDA that exports performance metrics describing the current state of items in the *sendmail* queue. |
| *pmdashping* | A PMDA that exports performance metrics for the availability and quality of service (response-time) for arbitrary shell commands. |
| *pmdasummary* | A PMDA that derives performance metrics values from values made available by other PMDAs. |
| *pmdatrace* | A PMDA that exports transaction performance metrics from application processes that use the *pcp_trace* library. |
| *pmdumplog* | The *pmdumplog* command displays selected state information, control data, and metric values from a PCP archive log created by *pmlogger*. |
| *pmlc* | The *pmlc* command is used to exercise 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. |
| *pmlogger* | The *pmlogger* command is used to create PCP archive logs of performance metrics over time. Many tools accept these PCP archive logs as alternative sources of metrics for retrospective analysis. |
| *pmlogextract* | The *pmlogextract* command reads one or more PCP archive logs and creates a temporally merged and reduced PCP archive log as output. |

*pmtrace* The *pmtrace* command provides a simple command-line interface to the trace PMDA and its associated *pcp_trace* library.

## Operational and Infrastructure Support

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

c*ron.** The *cron.pmcheck, cron.pmdaily,* and *cron.pmsnap* scripts are intended for periodic execution via *cron* to allow you to create a customized regime of administration and management for PCP archive log files and performance snapshots suitable for WWW publishing.

*dkmap* The *dkmap* tool creates a map of disk real estate usage.

*dkping* The *dkping* tool opens the named disk for reading and checks for a response.

*dkprobe* The *dkprobe* tool initializes disk performance metrics at boot time for some IRIX versions. It may be called from */etc/init.d/pcp*.

*memclaim* The *memclaim* tool allocates and holds physical memory, simulating a reduction in physical memory.

*pmbrand* The *pmbrand* tool manages the “branded” file of valid PCP licenses.

*pmdate* This tool displays the current date and/or time, with an optional offset.

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

*pmerr* The *pmerr* command translates PCP error codes into human-readable error messages.

*pmlaunch* The *pmlaunch* configuration directory 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.

*pmlock* The *pmlock* command attempts to acquire an exclusive lock by creating a file with a mode of 0.

*pmnewlog* The *pmnewlog* command is used to perform archive log rotation by stopping and restarting an instance of *pmlogger*.

| | |
|---|---|
| *pmnsadd* | The *pmnsadd* command adds a subtree of new names into a PMNS, as used by the components of PCP. |
| *pmnscomp* | The *pmnscomp* command compiles a PMNS in ASCII format into a more efficient binary representation. |
| *pmnsdel* | The *pmnsdel* command removes a subtree of names from a PMNS, as used by the components of the PCP. |
| *pmpost* | The *pmpost* command appends the text *message* to the end of the PCP notice board file (*/var/adm/pcplog/NOTICES*). |
| *pmrules* | The *pmrules* command provides a graphical user interface for instantiating and editing rules for *pmie*. |
| *pmstore* | The *pmstore* command re-initializes counters or assigns new values to metrics that act as control variables. The command changes the current values for the specified instances of a single performance metric. |

## Application and Agent Development

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

| | |
|---|---|
| *chkhelp* | The *chkhelp* tool checks the consistency of performance metrics help database files. |
| *dbpmda* | The *dbpmda* tool is an interactive debugger for PMDAs. This tool allows PMDA behavior to be exercised and tested. |
| *newhelp* | The *newhelp* tool generates the database files for one or more source files of PCP help text. |
| *PMAPI* | The Performance Metrics Application Programming Interface (PMAPI) defines a procedural interface for developing PCP client applications. |
| *pmclient* | The *pmclient* tool 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. |
| *pmgenmap* | The *pmgenmap* command is a program development tool that generates C declarations and *cpp* macros to aid the development of customized programs that use the facilities of PCP. |
| *PMDA* | A library used by many shipped PMDAs to communicate with a *pmcd* process. It can expedite the development of new and custom PMDAs. |

## Conceptual Foundations

The following sections provide a detailed overview of concepts that underpin PCP.

### Performance Metrics

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

The Performance Metrics Name Space (PMNS) provides a hierarchical classification of external metric names, and a mapping from external names to internal metric identifiers. See “Performance Metrics Name Space” on page 14 for a description of the PMNS.

### Performance Metric Instances

When performance metric values are returned to a requesting application, there may be more than one value instance for a particular metric; for example, independent counts for each CPU, process, disk, or local filesystem. Internal instance identifiers correspond one to one with external (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.

### 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 2.0, along with IRIX release 6.5, have been developed to be completely Year 2000 compliant.

## 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 following functional domains, each with an associated access method (see Figure 1-1):

- The IRIX kernel, including *sar* data structures, per-process resource consumption, network statistics, disk activity, or memory management instrumentation.
- A layered software service such as activity logs for a World Wide Web server, or an NNTP news server.
- A layered system product. For example, 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*.
- A DBMS. For example, the `V$` views and `bstat/estat` summaries for ORACLE, the `tbmonitor` statistics for INFORMIX, or the `sp_monitor` procedures for Sybase.
- External equipment such as network routers and bridges.
- An application program. For example, 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.

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

- 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

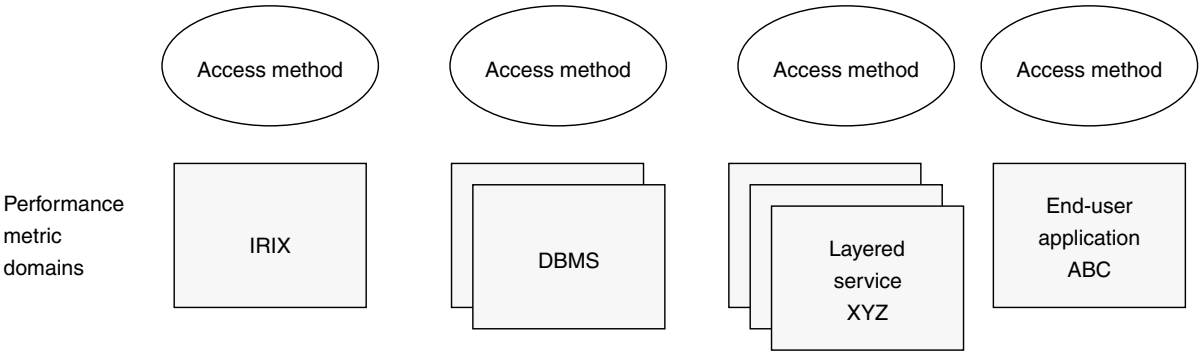


**Figure 1-1** Performance Metric Domains as Autonomous Collections of Data

We refer to each functional domain as a Performance Metrics Domain (PMD) and assume that PMDs are functionally, architecturally, and administratively independent and autonomous. Obviously the set of PMDs available on any host is variable, and changes with time as software and hardware are installed and removed.

The number of PMDs 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 PMDs must be both extensible at a particular host and distributed across a number of hosts.

Each PMD on a particular host must be assigned a unique PMD identifier. In practice, this means unique identifiers are assigned globally for each PMD type. For example, the same identifier would be used for the IRIX PMD on all hosts.

## Distributed Collection

The performance metrics collection architecture is distributed, in the sense that any performance tool may be executing remotely. However, a PMDA 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 1-2.

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.

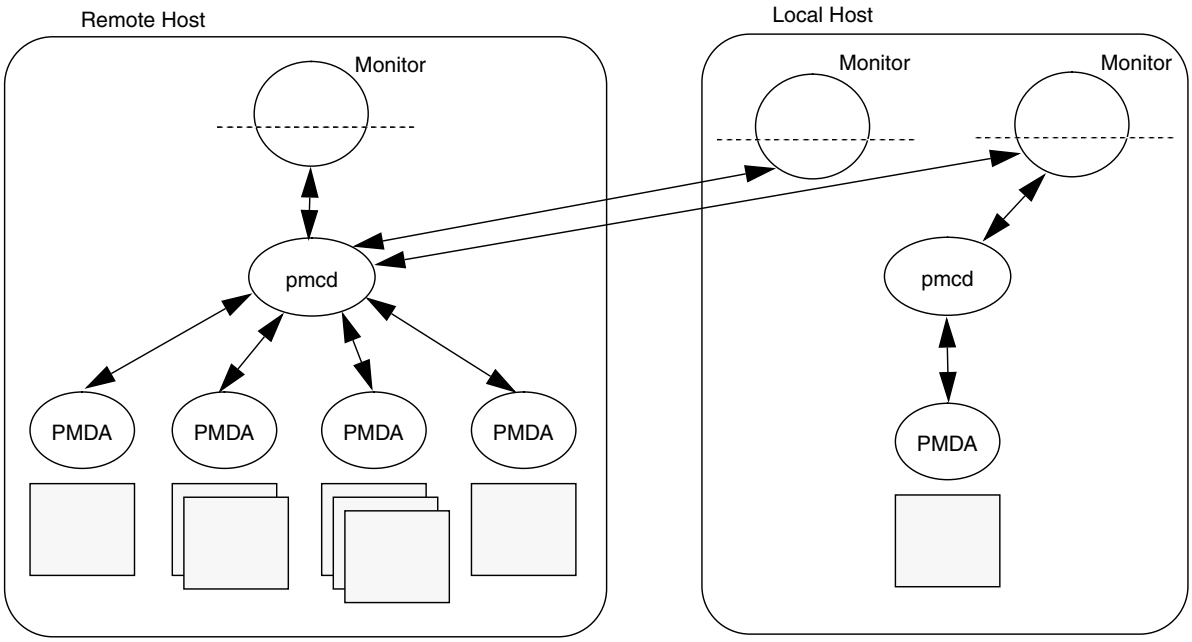


**Figure 1-2** Process Structure for Distributed Operation

The connections between monitor clients and *pmcd* processes are managed in *libpcp*, below the PMAPI level; see PMAPI(3). Connections between PMDAs and *pmcd* are managed by the PMDA routines; see PMDA(3). There can be multiple monitor clients and multiple PMDAs on the one host, but there may be at most one *pmcd* process.

## Performance Metrics Name Space

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

An external name space (the performance metrics name space, 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, in the small subsection of a PMNS shown in Figure 1-3, the following are valid names for performance metrics:

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

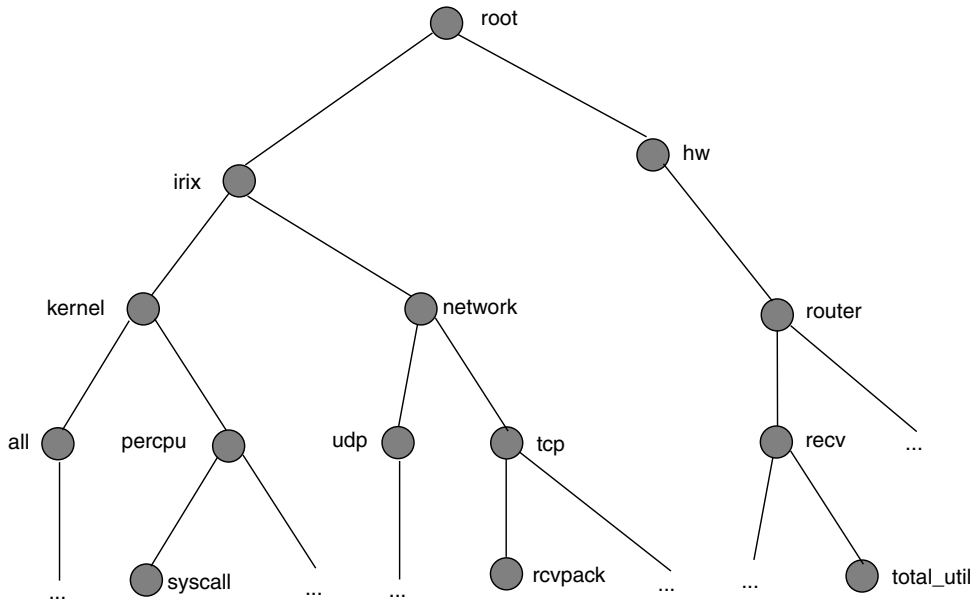


**Figure 1-3** Small Performance Metrics Name Space (PMNS)

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.

**Note:** For some low-end bundles based on PCP (such as WebMeter in the WebFORCE products), the default PMNS cannot be modified, nor can an alternate PMNS be created.

#### Distributed PMNS

In PCP 1.x releases, the PMNS was local to the application that referred to PCP metrics by name. As of 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. PCP release 2.0 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 PCP 1.x *pmlogger*, revert to using the local PMNS.

## 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 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, KB, or MB for the space dimension
- an indication if the metric may have one or many associated values
- short (and extended) help text describing the metric

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

## Values for Performance Metrics

There are two types of performance metrics, single valued and set valued, described in the following sections.

### Singular Performance Metrics

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

### Set-Valued Performance Metrics

Some performance metrics have a set of values or instances in each implementing PMD. 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 PMD 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 `irix.disk.dev.total` counts I/O operations for each disk spindle, and the associated instance domain contains one member for each disk spindle. On a system with five specific disks, one value would be associated with each of the external and internal instance identifier pairs shown in Table 1-1.

**Table 1-1** 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 PMD may dynamically establish the instances within an instance domain. For example, there may be one instance for the metric `irix.kernel.percpu.idle` on a workstation, but multiple instances on a multiprocessor server. Even more dynamic is `irix.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 PMDs to the applications that require this information. Applications may also choose to retrieve values for all instances of a performance metric, or some arbitrary subset of the available instances.

## Collector and Monitor Roles

Hosts supporting PCP services are broadly classified into two categories:

Collector—Hosts that have *pmcd* and one or more Performance Metric Domain Agents (PMDAs) running to collect and export performance metrics.

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

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

## Performance Metrics Collection System

Performance Co-Pilot 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 (PMDs)
- servicing incoming requests for local performance metric values and metadata from applications running either locally or on a remote system

## 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 1-4.

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, all PCP tools support live and retrospective monitoring with equal facility.

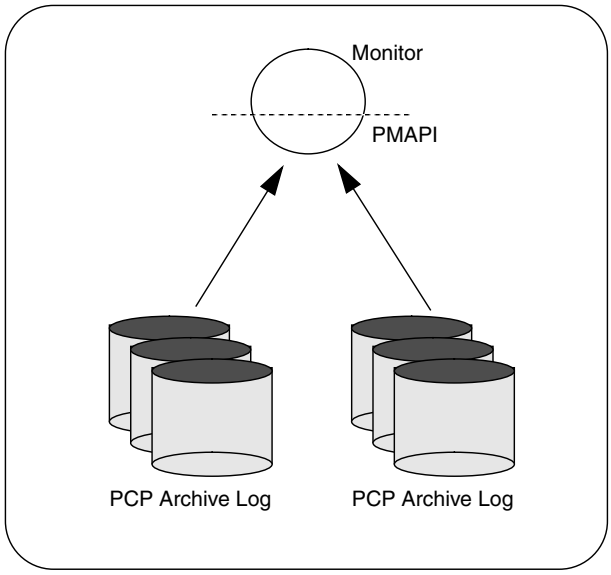


**Figure 1-4** Architecture for Retrospective Analysis

## 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 PMDA(3).
- 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 PMAPI(3).

*Chapter 2*

# Installing and Configuring Performance Co-Pilot

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

- "Product Structure" describes the main packages of PCP software and how they must be installed on each system.
- "Optional Software" on page 22 describes the additional options available for PCP.
- "License Constraints" on page 23 describes the licensing issues necessary to operate PCP in a distributed computing environment.
- "Performance Metrics Collector Daemon (PMCD)" on page 23 describes the fundamentals of maintaining the performance data collector.
- "Managing Optional PMDAs" on page 30 describes the basics of installing a new Performance Metric Domain Agent to collect metric data and pass it to the PMCD.
- "Troubleshooting" on page 32 offers advice on problems involving the PMCD.

## Product Structure

In a typical deployment, Performance Co-Pilot 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 *inst* or *swmgr*:

- 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 collector 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 sgihelp(1).

- collector—No additional installation is required because the performance monitor collector 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.
- gifts— 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, reference 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* or *grelnotes* commands.

## Optional Software

The capabilities of your 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:

- Several common database packages. These optional products provide customized PMDAs (performance metric domain agents) and visual analysis tools for the ORACLE, Sybase, and INFORMIX database systems.
- World Wide Web server performance monitoring.
- Platform-specific extensions for Silicon Graphics array and IRIS FailSafe products.

In addition, WebMeter is a bundled package that supports reduced PCP functionality and services for World Wide Web servers.

## License Constraints

On PCP monitoring systems, all of the display, visualization, and automated reasoning tools are licensed using “nodelocked” FLEXlm licenses. On PCP collector 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 PMDAs or *pmlogger*) may be installed and executed without license constraints.

Some of the PCP maintenance tools for updating the PMNS, interrogating the PMCS, dumping an archive log, and so on, are not constrained by any license restrictions.

### 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
```

## Performance Metrics Collector Daemon (PMCD)

On each PCP collector system, you must be certain that the *pmcd* daemon is running. This daemon co-ordinates the gathering and exporting of performance statistics in response to requests from the PCP monitoring tools.

### Starting and Stopping the PMCD Daemon

To start the daemon, enter the following commands as root on each PCP collector 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 collector system, enter the command

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

## Restarting an Unresponsive PMCD Daemon

Often, if a daemon is not responding on a PCP collector 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 collector 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) reference 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.

## 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 environment variable PCPLOGDIR before running */etc/init.d/pcp* allows the file to be relocated.

## 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.

### 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 accepted options are listed below:

**-f** This option 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* This option 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* This option specifies the amount of time, in seconds, before PMCD times out on 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* This option specifies whether connection and PDU tracing are turned on for debugging purposes.

See the pmcd(1) reference 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
```

### 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 Metric 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) reference page.

With standard operation of Performance Co-Pilot (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 DSO or 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 pathname*

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*. |

#### Controlling Access to PMCD With pmcd.conf

There is an option extension you can place 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:

- Hostnames 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 wildcard 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 | This operation 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 | This operation 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) reference page.

## Managing Optional PMDAs

Some PMDAs (Performance Metrics Domain Agents) shipped with 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.

### 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.

#### Installation on a PCP Collector 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 PMD (Performance Metrics Domain) 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 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. The sample output below is illustrative of the interactions:

```
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
```

### 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.

#### Removal on a PCP Collector 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 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
   ```

## Troubleshooting

The following sections offer troubleshooting advice.

Advice for troubleshooting the archive logging system is provided in Chapter 7, "Archive Logging."

## Troubleshooting the Performance Metrics Name Space (PMNS)

To display the PMNS, use the *pminfo* command; see pminfo(1).

The PMNS at the collector 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.

## Missing and Incomplete Values for Performance Metrics

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

### Metric Values Not Available

The following symptom has a known cause and resolution:

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

Cause: This can occur as a consequence of changes in the installation of modules (for example, a DBMS or an 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 collector host supports the metric.

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

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

## Troubleshooting IRIX Metrics and the PMCD

The following issues involve IRIX and the PMCD:

- no IRIX metrics available
- cannot connect to remote PMCD
- PMCD not reconfiguring after hangup

### 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 form

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

means that PMCD cannot access */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.

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

### 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
```

### 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 below:

```
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, there has been a problem 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 re-establish 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 states, 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 PCPIntro(3) for a description of how to override the default PMCD port assignment using the environment variable PMCD_PORT.

*Chapter 3*

# Common Conventions and Arguments

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. These are the major sections in this chapter:

- "PerfTools Icon Catalog" on page 38 shows a picture of the PerfTools icons.
- "Alternate Metric Source Options" on page 39 details some basic standards used in the development of PCP tools.
- "General PCP Tool Options" on page 40 details other options to use with PCP tools.
- "Time Duration and Control" on page 41 describes the time control dialog and time-related command-line options available for use with PCP tools.
- "PCP Environment Variables" on page 48 describes the environment variables supported by PCP tools.
- "Running PCP Tools Through a Firewall" on page 51 describes how to execute PCP tools that must retrieve performance data from *pmcd* on the other side of a TCP/IP security firewall.
- "Transient Problems With Performance Metric Values" on page 51 covers some uncommon scenarios that may compromise performance metric integrity over the short term.

Many of the utilities provided with Performance Co-Pilot (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 reference 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.

## 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 3-1.

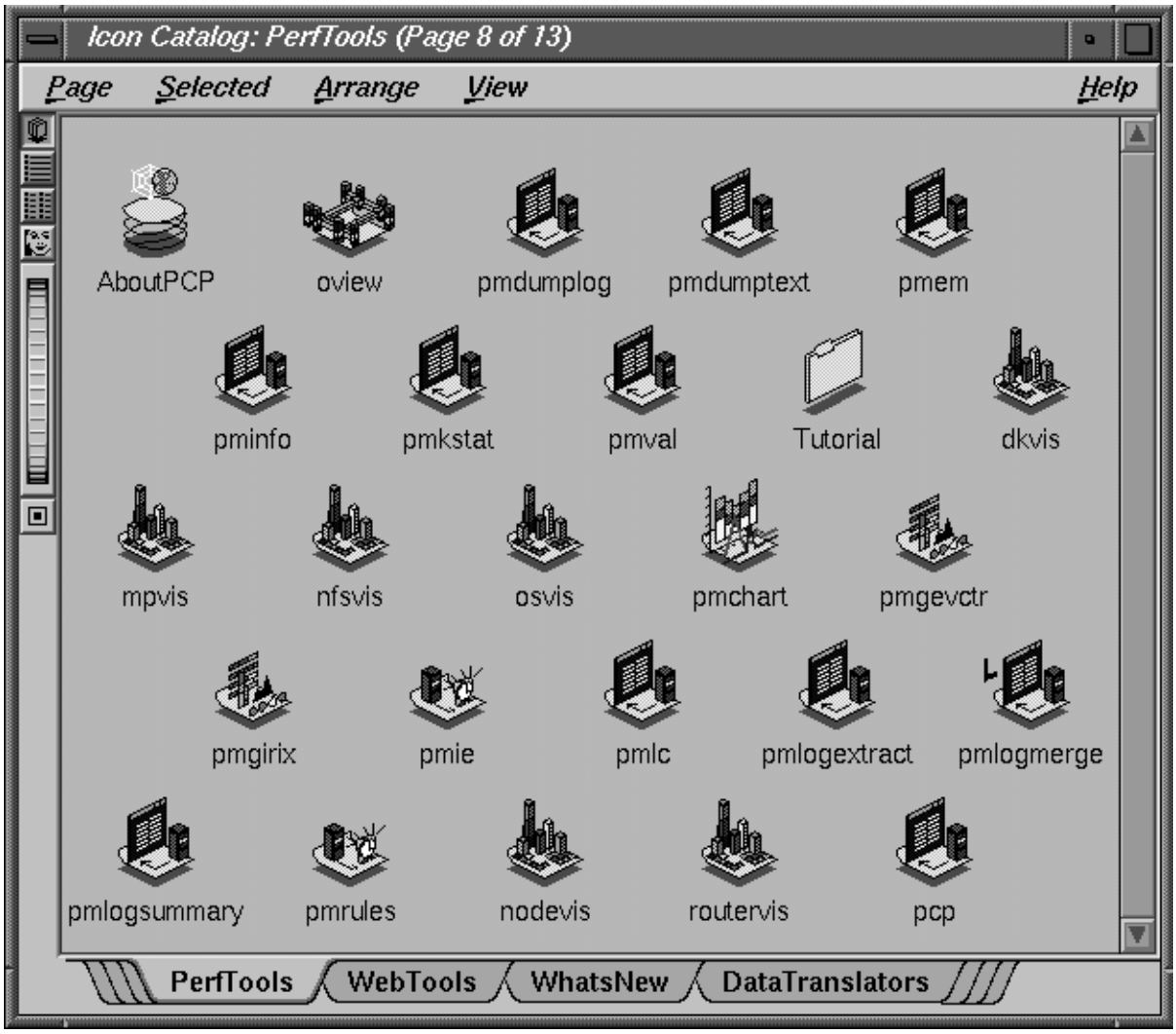


**Figure 3-1** PerfTools Icon Catalog Group

## Alternate Metric Source Options

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

### 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.

### 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, "Archive Logging."

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 3-1.

**Table 3-1** 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.

## 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.

### 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 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 Graphical Tools such as *dkvis*, *nfsvis*, and *pmview*. |
| */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/doc* | PCP WebMeter documentation shipped in HTML format, suitable for display with some World Wide Web browser. |
| */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 files and programs and the PCP Tutorial. |

*/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.

### 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 2.0 avoids most requirements for an alternate PMNS, because clients' PMNS operations are supported at the 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) reference pages for details of PMNS structure and creation.

## 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.

### 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) reference page.

## 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

```
-S 30min
```

retrieves the first sample of data at exactly 30 minutes from the beginning of a PCP archive.

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

```
-S -1hour
```

retrieves the first sample exactly one hour preceding the last sample in a PCP archive.

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,

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

specifies the date and time to be used. 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) reference 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 realtime, 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 **-S** 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.

## Time Zone Options

All utilities that report time of day use the local time zone by default.

**-z** This option 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* This option may be used to set the TZ variable to a time zone string, as defined in environ(5), for example, `-Z UTC` for universal time.

## PCP Live Time Control

The *pmtime* dialog 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 reference pages for pmview(1), pmchart(1), oview(1), or pmtime(1).

If the PCP tool is displaying performance metrics from a real-time source, the *pmtime* dialog looks similar to that shown in Figure 3-2.

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 option menu (labelled Seconds in Figure 3-2).


**Figure 3-2** pmtime Live Time Control Dialog

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.

## Creating a PCP Archive

The ability to start and stop recording of performance activity is available from the *pmchart*, *pmview*, and *oview* windows. See "Creating a PCP Archive From a pmchart Session" on page 64 for information about the *pmchart* interface.

Alternatively use *pmlogger* directly, as described in Chapter 7, "Archive Logging."

## 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.

When a PCP tool is displaying performance metrics from a PCP archive log, and the *pmtime* dialog is exposed, it looks similar to that shown in Figure 3-3.

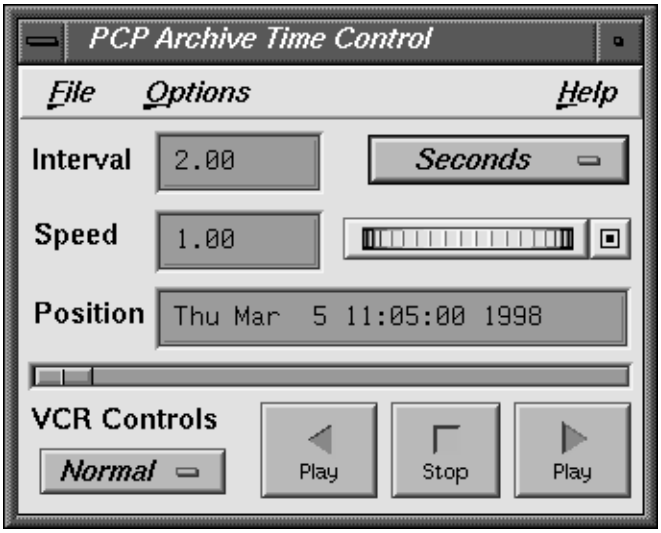


**Figure 3-3** pmtime Archive Time Control Dialog

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 realtime.
- 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, Rewind, FastFwd, or Stop), or by modifying the Position text box (and pressing Enter), or by moving the slider below the Position text box.
- The VCR motion buttons allow time to be advanced forward or backward, or stopped.

The menus of *pmtime* ArchiveTime Control provide the following additional features:

#### File Menu

Hide option
: Hide the dialog; the PCP tools provide their own menu options or time control icon that may be used to re-expose the *pmtime* dialog.

#### Options Menu

Timezone option
: Select 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 option menu. The default time zone is that of the local host where the tool is being run.

Show Bounds... option
: Expose the Archive Time Bounds dialog, which shows the current time window that defines the earliest and latest time for which performance may be displayed from the current archives.


**Figure 3-4** pmtime Archive Time Bounds Dialog

Detail option — For output fields, selectively include or exclude the year in the date or milliseconds in time. The year is shown by default, milliseconds are not.

## PCP Environment Variables

The following environment variables are recognized by PCP (these definitions are also available on the PCPIntro(1) reference 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 environment variable PCP_COUNTER_WRAP indicates that all such cases of a decreasing counter should be treated as a counter overflow; and hence 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 directory */var/adm/pcplog*. Setting the variable PCP_LOGDIR overrides the default directory. If PCP_LOGDIR is unset, the variable PCPLOGDIR 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 pmprintf(3). 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 xconfirm(1). 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 pmdatrace(3).

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 pmdatrace(3).

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 pmdatrace(3).

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 is the TCP/IP port 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 their 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_LOCAL_PROC
: If set, then a context established with type PM_CONTEXT_LOCAL has access to the *proc* PMDA and can retrieve performance metrics about individual processes.

PMDA_LOCAL_SAMPLE
: If set, then a context established with type PM_CONTEXT_LOCAL has access to the *sample* PMDA if this optional PMDA has been installed locally.

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 pmlaunch(5).

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.

## Running PCP Tools Through a Firewall

In some production environments, the 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 pmsocks(1). 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 environment variable PMCD_PORT) for connections to *pmcd* and a finite range of consecutive port numbers starting at 4330 (or the value of environment variable PMLOGGER_PORT) to allow *pmlc* connections to *pmlogger* instances.

### The pmsocks Command

The *pmsocks* command and its related files and scripts allow Performance Co-Pilot (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 this is the file where the network-specific information must be set to correspond with your network. Complete information on this customization can be found in the pmsocks(1) reference page.

## 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 wrap-around or time skew, described below. These conditions result from design decisions that are biased in favor of light-weight 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.

## Performance Metric Wrap-Around

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 wrap around 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.

## 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 timestamps 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 timed(1M).

*Chapter 4*

# Monitoring System Performance

This chapter describes the performance monitoring tools available in PCP. This product provides a group of commands and tools for measuring system performance. Each tool is described completely by its own reference page. The reference pages are accessible through the *man* command. For example, the reference page for the tool *pmchart* is viewed by entering the command

```
man pmchart
```

The following major sections are covered in this chapter:

- "The pmchart Tool" on page 54 describes a useful charting tool that graphically monitors system performance.
- "The pmgadgets Command" on page 69 presents a graphical tool that displays system performance in a small area.
- "The pmkstat Command" on page 73 discusses a utility that provides a periodic one-line summary of system performance.
- "The pmval Command" on page 75 discusses a utility that shows the current values for named performance metrics.
- "The pmem Command" on page 77 discusses a utility that reports per-process memory usage statistics.
- "The pminfo Command" on page 78 describes a utility that displays information about performance metrics.
- "Changing Metric Values With pmstore" on page 81 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, "System Performance Visualization Tools" and Chapter 6, "Performance Metrics Inference Engine."

The following sections describe the various 2D and text-based PCP tools used to monitor local or remote system performance.

## 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. When you enter the command

```
pmchart
```

you see the window shown in Figure 4-1.

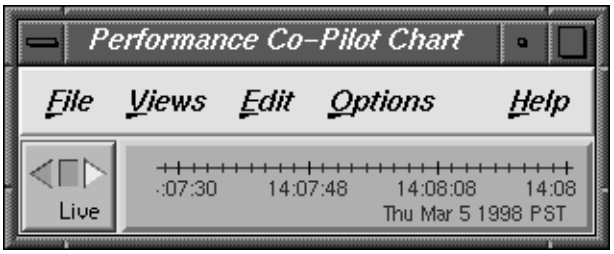


**Figure 4-1** pmchart Window

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 reference page for *pmchart* explains how to configure charts based on performance metrics, using either the View Selector option of the Views menu or the New Chart option of the Edit menu. Once charts have been configured and applied, the charts are placed in an expanded window, as shown in Figure 4-2.

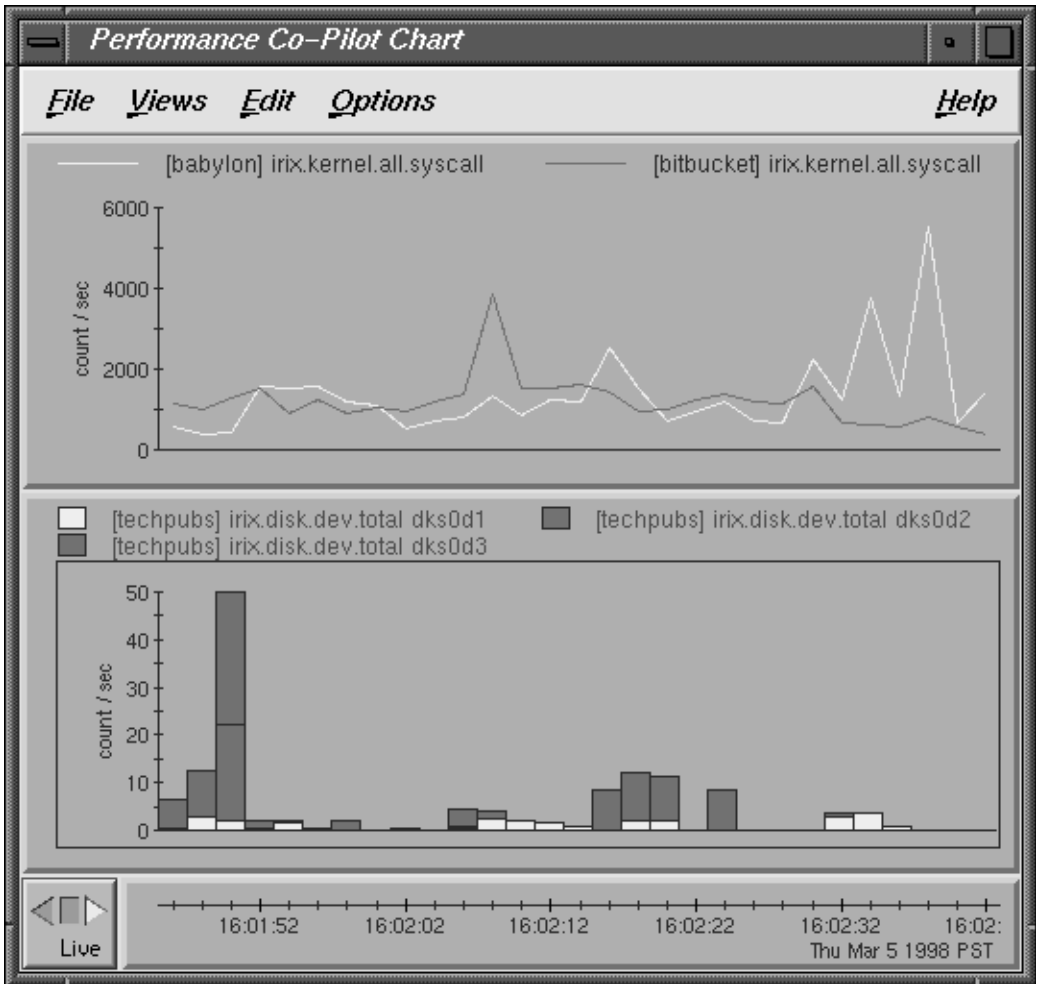


**Figure 4-2** Two Charts and Metrics From Three Hosts in pmchart

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.

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 post-mortem 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 "Time Duration and Control" on page 41.

## Mouse Controls

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

| | |
|---|---|
| Left button | 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 button | The middle mouse button is unused. |
| Right button | 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. |

## pmchart View Selection

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 View Selector option in the Views menu launches a dialog box as shown in Figure 4-3, which may be used 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. |

The View Selection dialog looks similar to Figure 4-3.

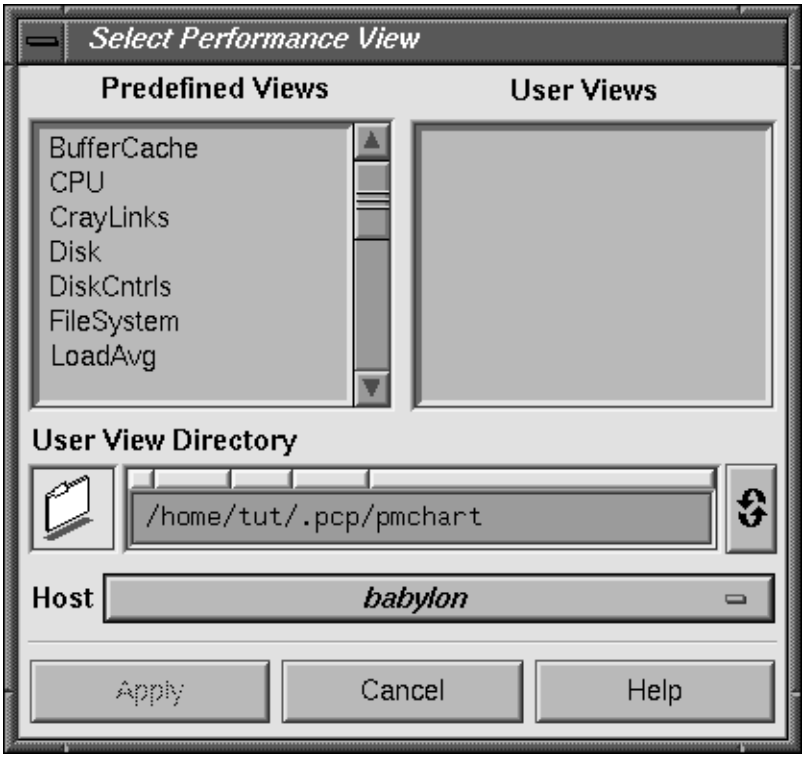


**Figure 4-3** pmchart View Selection Dialog

You can create your own custom views using the metric selection facilities described below. Then use the View Save option in the Views menu to save your setup as a custom view, displayed in the User Views list.

## pmchart Metric Selection

The Metric Selection dialog window allows interactive navigation of the PMNS (Performance Metrics Name Space) to create new chart configurations. The dialog gives you the ability to choose metrics, display information about performance metrics, change the current host or archive, select metric instances, and then plot metric values on a common time axis. You obtain this dialog by selecting the New Chart option from the Edit menu of *pmchart*. This dialog is shown in Figure 4-4.

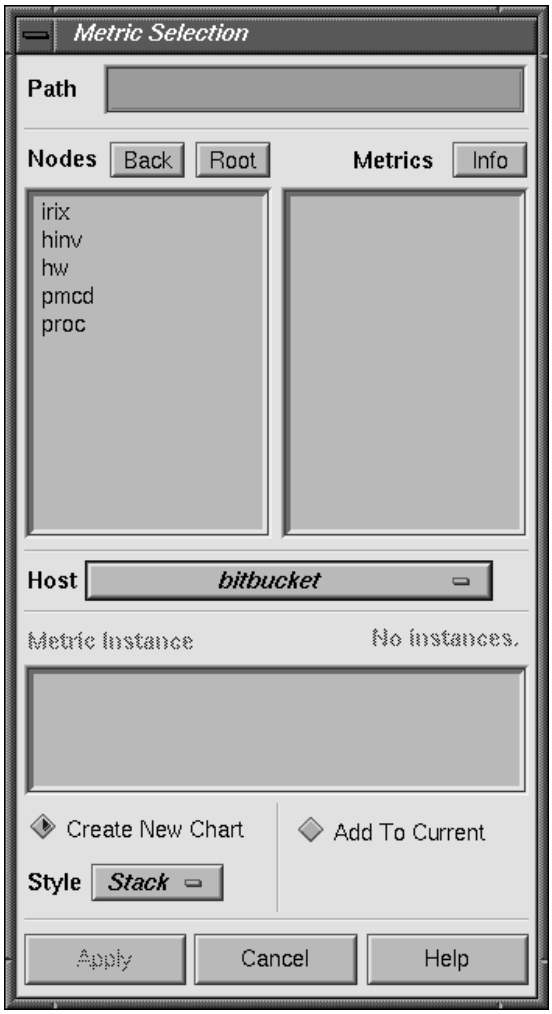


**Figure 4-4** Metric Selection Dialog

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 to select the metrics you need. For example, if you enter the path *irix.network.interface*, the window changes dynamically, as shown in Figure 4-5.

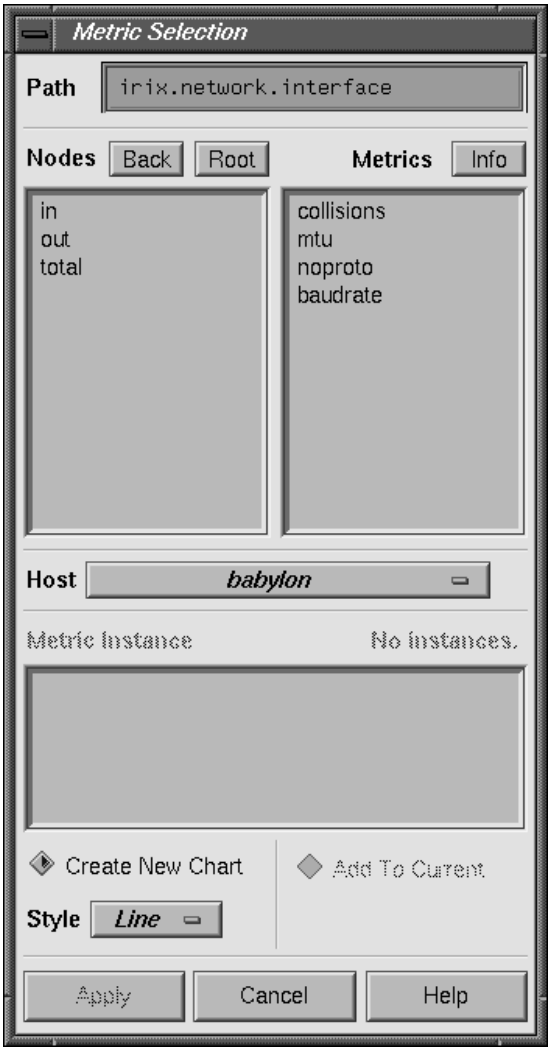


**Figure 4-5** Further Metric Selection

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 4-6.

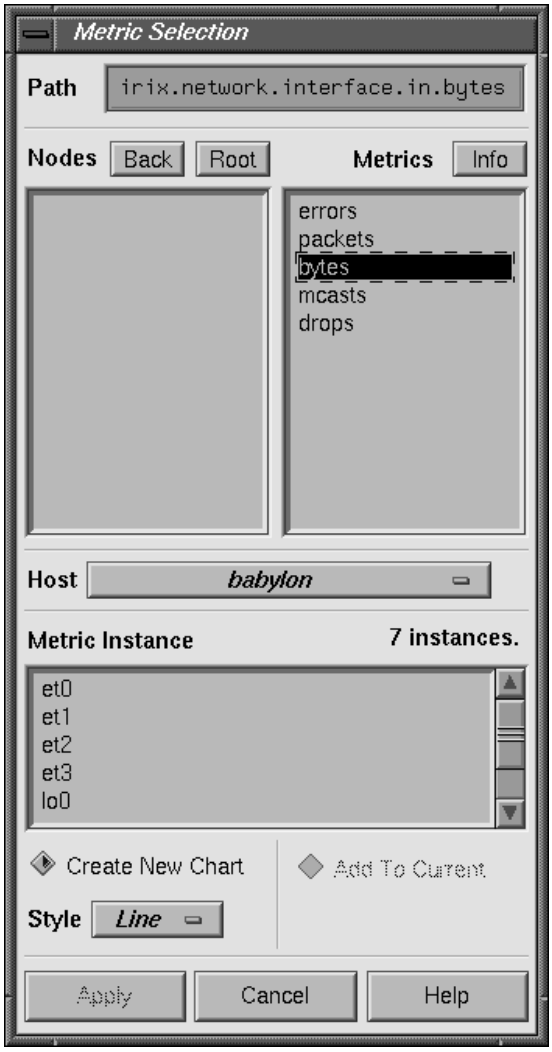


**Figure 4-6** Selecting a Leaf Node in the PMNS (Performance Metric)

Once a metric has been selected, the Info button in the Metric Selection dialog launches the Metric Information dialog, as shown in Figure 4-7. 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.

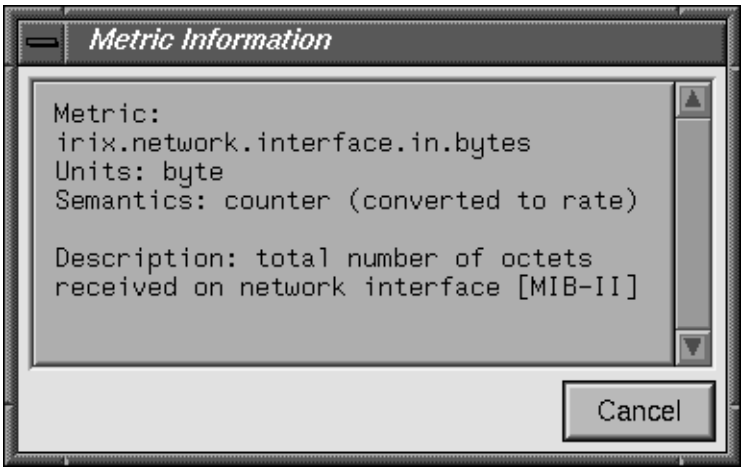


**Figure 4-7** Metric Information Dialog

Finally, you may have to select from several instances of a metric. In the example shown in Figure 4-7, 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 4-8.

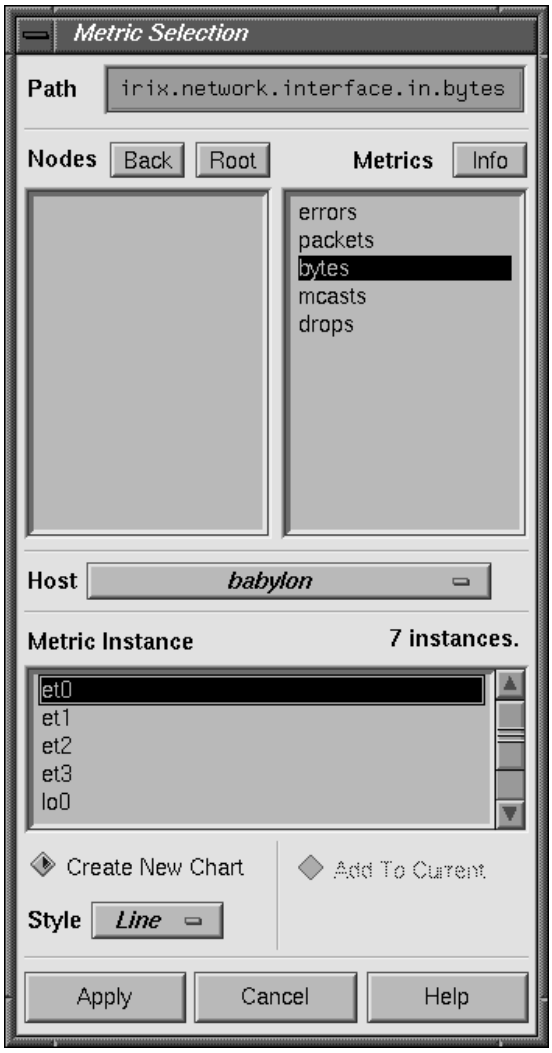


**Figure 4-8** Selecting a Metric Instance

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.

## 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 re-started (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 the section "PCP Archive Folios" on page 147). 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 4-9.

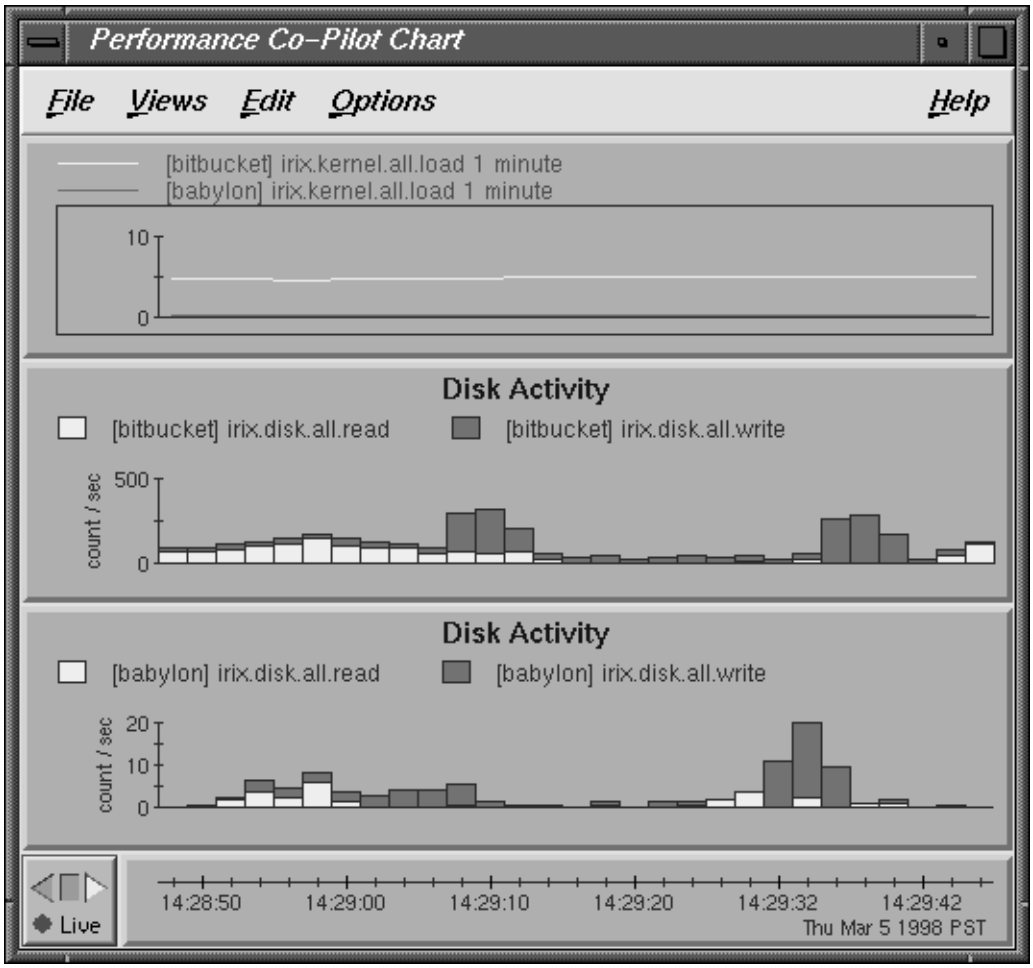


**Figure 4-9** pmchart Display When Recording

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, a dialog similar to that shown in Figure 4-10 appears to remind you where the archive folio was created, and to confirm that recording should be terminated.

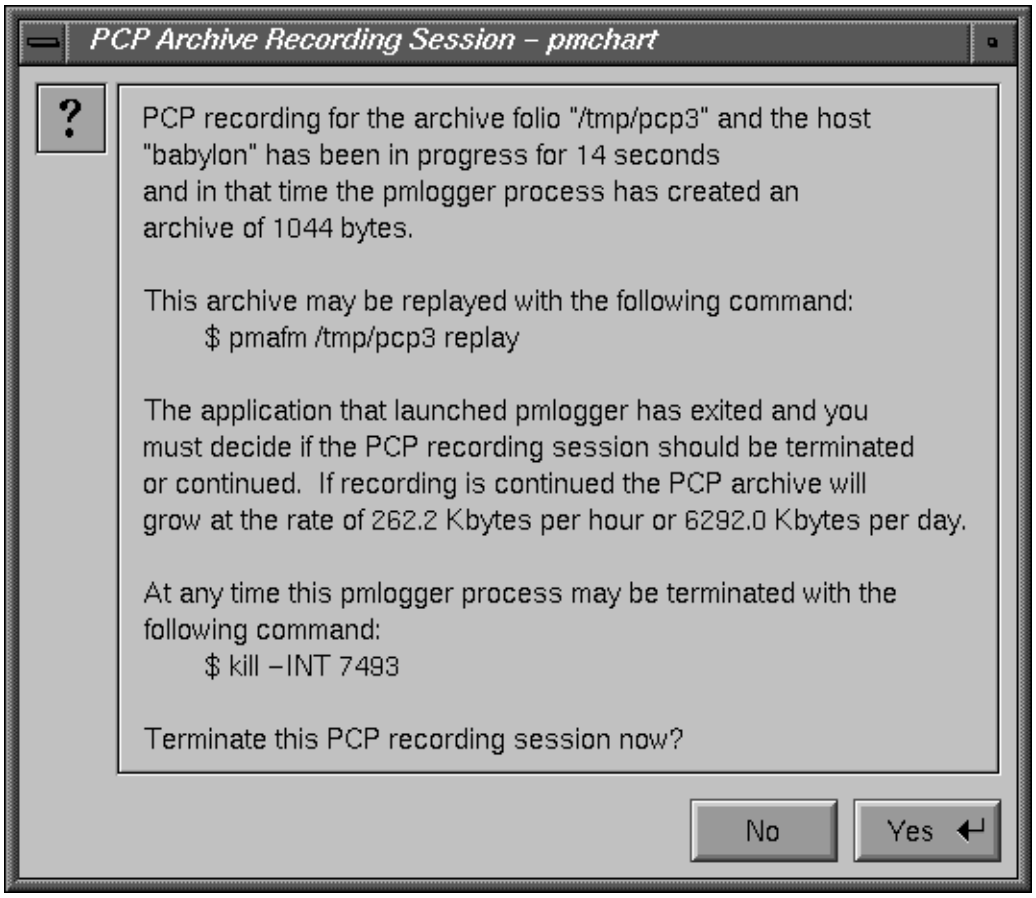


**Figure 4-10** pmchart Stop Recording Dialog

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.

## 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 file *$HOME/.xrdb*:

```
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
```

## Other Chart Customizations

The *pmchart* Edit menu provides options and a dialog that may be used to change and customize the display as follows:

- Visible Data Points: a slider to change the number of values along the time axis.
- Chart Style: choose from line, bar, stacked bar, area plot, and utilization.
- 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: either all charts, a complete chart, or individual plots from a chart.

## Time Control

The Options menu provides access to the PCP Time Control Dialog (as described in the section "Time Duration and Control" on page 41).

The Show Time Control option 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).

The New Time Control option detaches *pmchart* from the controlling *pmtime* instance and launches a new *pmtime* instance, initially dedicated to this *pmchart*.

The Launch New Pmchart option starts a new *pmchart*, with shared *pmtime* control.

## 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 *cron.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/cron.pmsnap*
: The *cron.pmsnap* script is designed to be periodically run by *cron* to process the control file */var/pcp/config/pmsnap/control* and generate snapshot images according to the specifications therein. The reference page for *cron.pmsnap* 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 *cron.pmsnap* are in the reference page. There is also a verbose comment at the head of the control file. The pmchart(1) reference page is also useful.

### 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
```

## 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, *pmgirix* generates a configuration file for various IRIX performance metrics and feeds it to *pmgadgets*. The resulting display depends on the host configuration, but the following is representative of a system with four CPUs, eleven disks on three controllers, and four network interfaces.

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

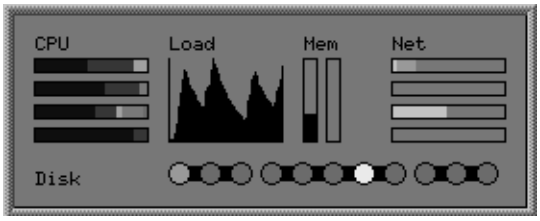


**Figure 4-11** Representative pmgadgets Display Using pmgirix

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

- _bar: A gadget that displays a single performance metric value as a rectangle that is filled from left to right or bottom to top in proportion to the ratio of the metric's value to some maximum.
- _multibar: 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: A gadget that displays a simple *xload* style strip chart of a performance metric's values over time.
- _led: A circular gadget whose color is modulated using the value of a single performance metric.
- _line: A solid rectangle, not modulated by any performance metric, useful for highlighting connectivity between gadgets.
- _label: Textual annotation in the display.

- _actions: 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: Lists of X11 color specifications.
- _legend: 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 4-1 includes CPU, disk, and load average information from two hosts, and produces a display like the one shown in Figure 4-12.

**Example 4-1** Sample 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:irix.kernel.all.cpu.user
        moomba:irix.kernel.all.cpu.sys
        moomba:irix.kernel.all.cpu.intr
        moomba:irix.kernel.all.cpu.wait.total
        moomba:irix.kernel.all.cpu.idle
    )
    _maximum 0.0
    _colourlist cpuColours

_bargraph 40 5 25 20
    _metric moomba:irix.kernel.all.load["1 minute"]
    _max 1.0

_led 12 16 6 6
    _metric moomba:irix.disk.all.read _legend diskLegend
```

```
_led 25 16 6 6
    _metric moomba:irix.disk.all.write _legend diskLegend

# host gonzo
_label 70 39 "gonzo"

_multibar 5 32 30 6
    _metrics (
        gonzo:irix.kernel.all.cpu.user
        gonzo:irix.kernel.all.cpu.sys
        gonzo:irix.kernel.all.cpu.intr
        gonzo:irix.kernel.all.cpu.wait.total
        gonzo:irix.kernel.all.cpu.idle
    )
    _maximum 0.0
    _colourlist cpuColours

_bargraph 40 32 25 20
    _metric gonzo:irix.kernel.all.load["1 minute"]
    _max 1.0

_led 12 43 6 6
    _metric gonzo:irix.disk.all.read _legend diskLegend

_led 25 43 6 6
    _metric gonzo:irix.disk.all.write _legend diskLegend
```

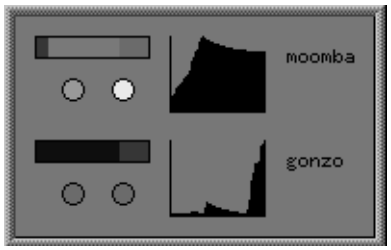


**Figure 4-12** Customized pmgadgets Display

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, to track the current values of the performance metrics that are associated with the gadget. For example, see Figure 4-13.

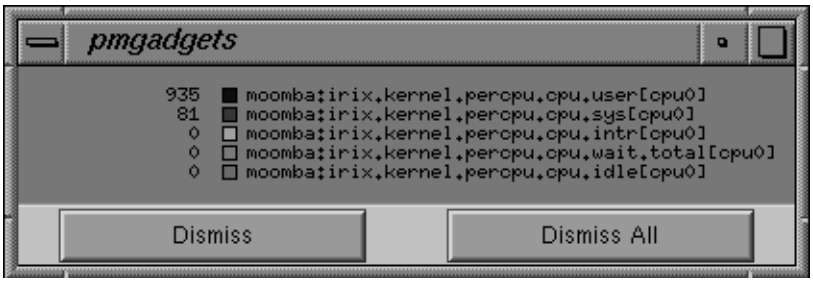


**Figure 4-13** pmgadgets Information Dialog

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

## 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 metrics and archive logs are interchangeable.

For example, the following command uses the PCP archive log *foo* and the timezone 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) reference page.

## The pmdumptext Command

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

```
pmdumptext -imu -h serv -f '%H:%M:%S' irix.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 pmdumptext(1) for more information.

## 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 irix.kernel.percpu.cpu.idle
```

```
metric:    irix.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 irix.disk.dev.read
```

```
metric:    irix.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 the section "Time Duration and Control" on page 41) 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) reference page. There are annotated examples of the use of *pmval* in the PCP Tutorial.

## The pmem Command

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

Both virtual size and pro-rated 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 | Pro-rated 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 | Pro-rated 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 | Pro-rated 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) reference page.

## The pminfo Command

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

The **-T** option is extremely useful—it provides help text about performance metrics:

```
pminfo -T irix.mem.util.fs_dirty
```

```
irix.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 irix.network.interface.total.bytes
```

```
hinv.ncpu
irix.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
```

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

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

```
pminfo -d proc.nprocs irix.disk.dev.read irix.filesys.free
```

```
proc.nprocs
    Data Type: 32-bit int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: instant  Units: none
```

```
irix.disk.dev.read
    Data Type: 32-bit unsigned int  InDom: 1.2 0x400002
    Semantics: counter  Units: count
irix.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 irix.filesys.mountdir
```

```
irix.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) reference page. There are examples of the use of *pminfo* in the PCP Tutorial.

## Changing Metric Values With pmstore

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 pmstore(1).

*Chapter 5*

# System Performance Visualization Tools

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 reference page, accessible through the *man* command. For example, the reference page for *pmview* can be viewed by giving the following command:

```
man pmview
```

The following major sections are covered in this chapter:

- "Overview of Visualization Tools" on page 84 provides background motivation and places the current chapter in the context of other PCP tools.
- "The dkvis Disk Visualization Tool" on page 85 describes the graphical disk activity visualization tool, *dkvis*.
- "The mpvis Processor Visualization Tool" on page 88 describes the graphical multi-processor visualization and comparison tool, *mpvis*.
- "The osvis System Visualization Tool" on page 90 describes the operating system activity visualization tool, *osvis*.
- "The nfsvis NFS Activity Visualization Tool" on page 92 describes the graphical NFS activity visualization tool, *nfsvis*.
- "The oview Origin Visualization Tool" on page 94 describes the Origin visualization tool, *oview*.
- "The pmview Tool" on page 95 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) reference pages for information about these *pmview*-based tools.

## 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, "Performance Metrics Inference Engine," 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-dimensional 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 Silicon Graphics 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-dimensional models for
  - 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-dimensional 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*.

- Visual representation of physical memory allocation on a per-process or per-region-per-process basis, *memvis*.

  **Note:** *memvis* is provided for PCP on IRIX 5.3 platforms. Later IRIX versions include *gmemusage*, which is functionally equivalent to *memvis*.

- Icon-sized stripcharts, meters, and indicator LEDs may be combined into a desktop control and indicator panel using *pmgadgets*; see "The pmgadgets Command" on page 69. The *pmgirix* 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* is a sample multi-level performance visualization for Oracle parallel server configurations. It 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.

## 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 window similar to the one shown in Figure 5-1.

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 5-1, the disks in the row labelled *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 "The pmview Tool" on page 95. 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 thumbwheel in the *pmview* viewer.

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

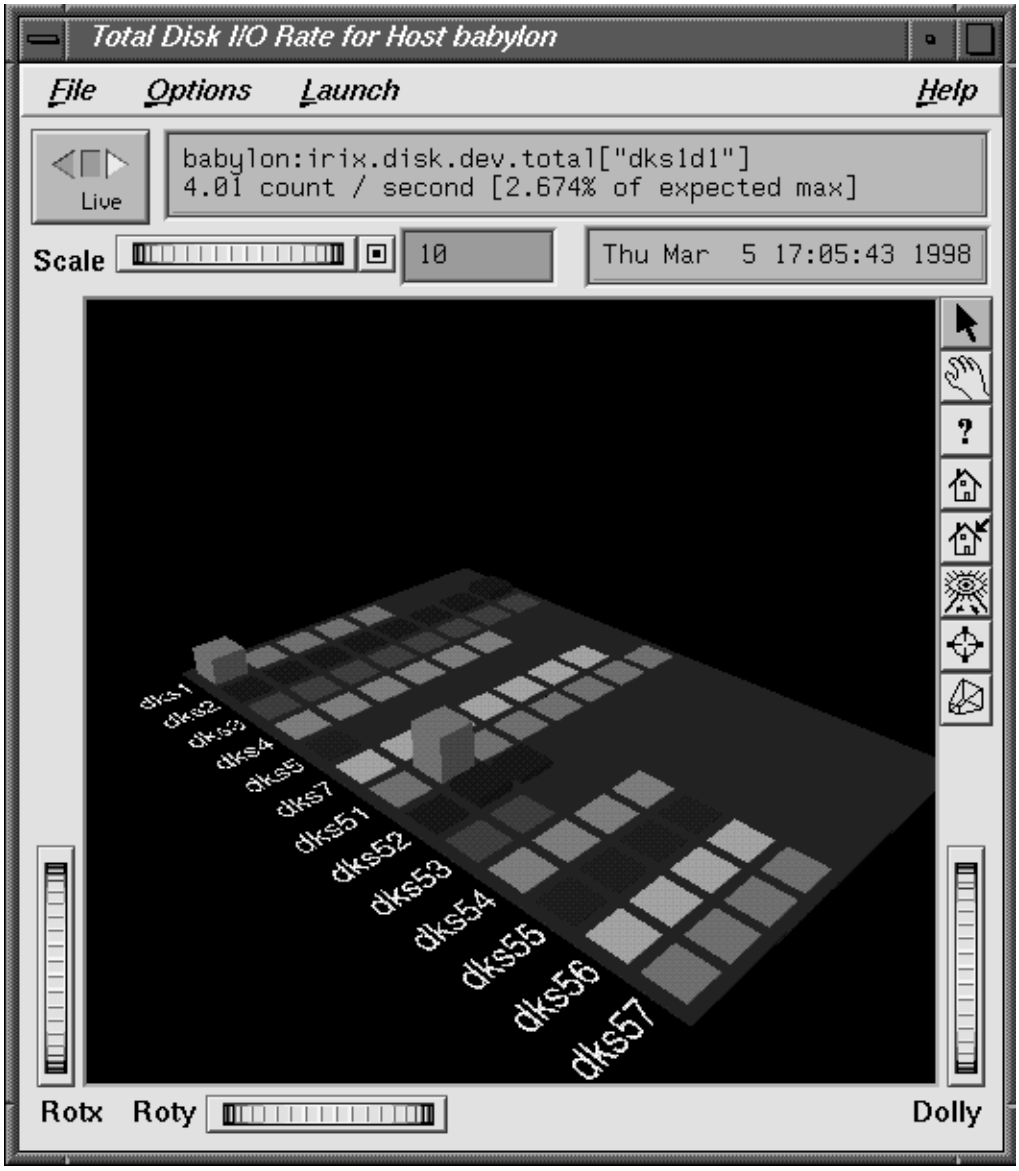


**Figure 5-1** dkvis Window

## 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 window similar to the one shown in Figure 5-2.

Figure 5-2 shows *mpvis* monitoring a machine with four CPUs. Notice in the figure that CPU1 is spending 28% of its time processing user code and slightly more time executing system code. The other CPUs are mostly idle with a small proportion of time spent waiting for I/O.

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 *dkvis* implementation uses the generalized 3D performance viewer *pmview* as described in “The pmview Tool” on page 95. 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) reference page. The PCP Tutorial contains additional examples on the use of *mpvis*.

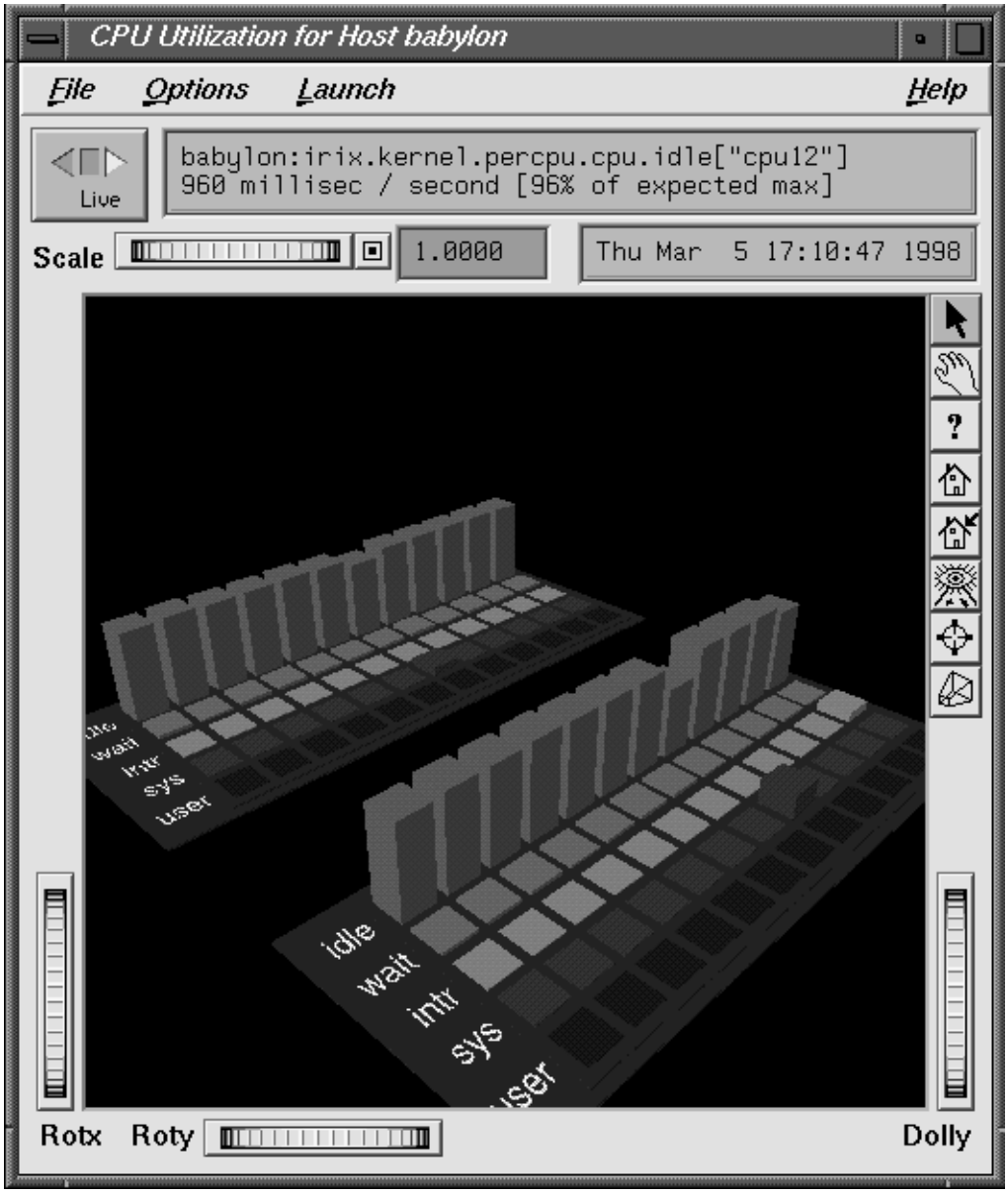


**Figure 5-2** mpvis Window

## The osvis System Visualization Tool

The *osvis* tool displays an 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 5-3 shows a sample *osvis* display.

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. |
| memory | The stack shows memory utilization by breaking down real memory into kernel, file system and user usage. The memory utilization metrics (*irix.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. |
| 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 in | The two rows of bars show the input byte rate and the input packet rate for each network interface, except loopback and slip interfaces. |
| network out | The two rows of bars show the output byte rate and the packet rate for each network interface, except loopback and slip interfaces. |

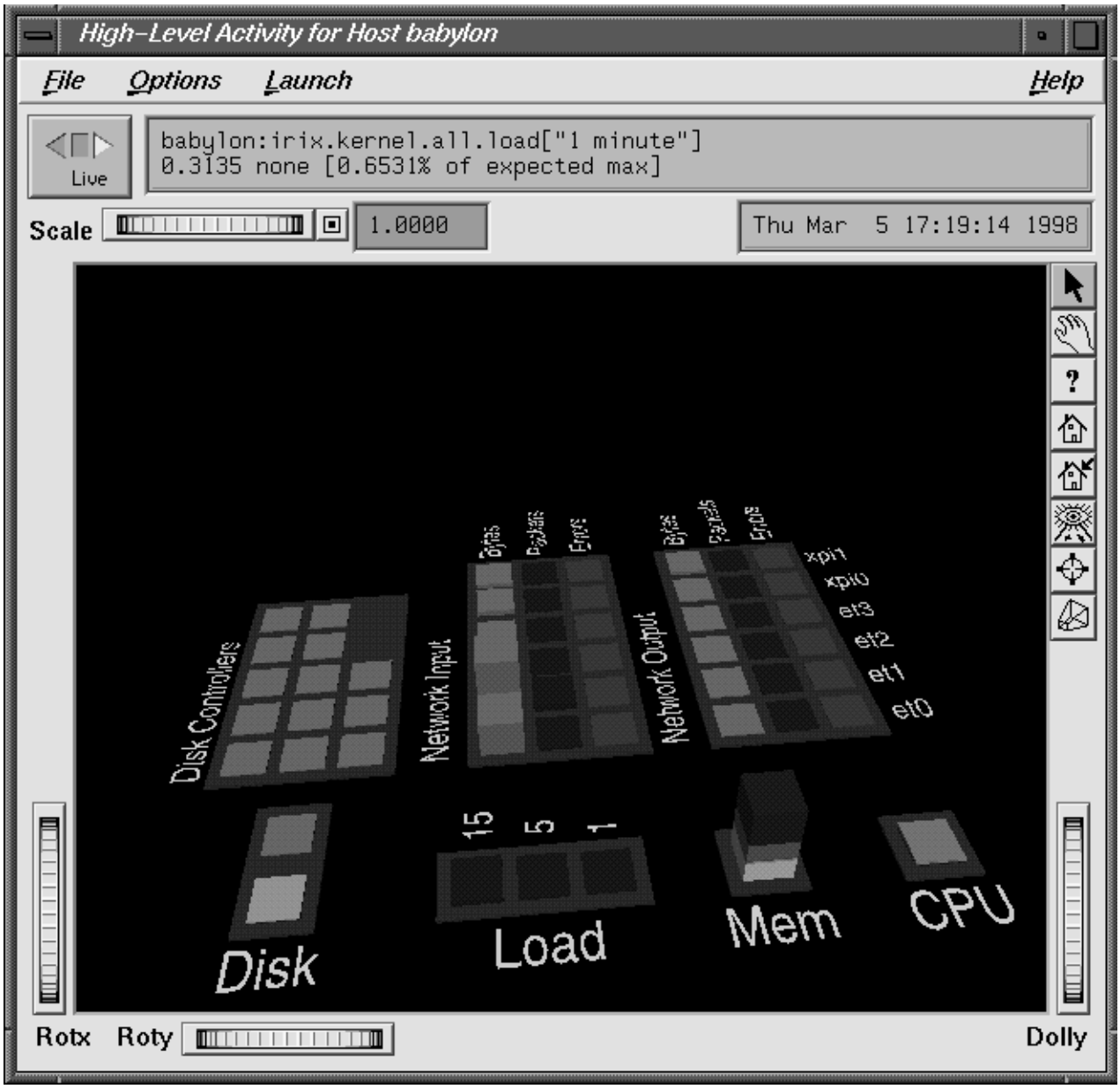


**Figure 5-3** osvis Window

## 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 window similar to the one shown in Figure 5-4.

The statistics are broken into two groups: server statistics (requests from other machines for the NFS server on the machine being monitored) and client statistics (requests made by the monitored machine to NFS servers on other machines). 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 “The pmview Tool” on page 95. 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) reference page. The PCP tutorial contains additional examples on the use of *nfsvis*.

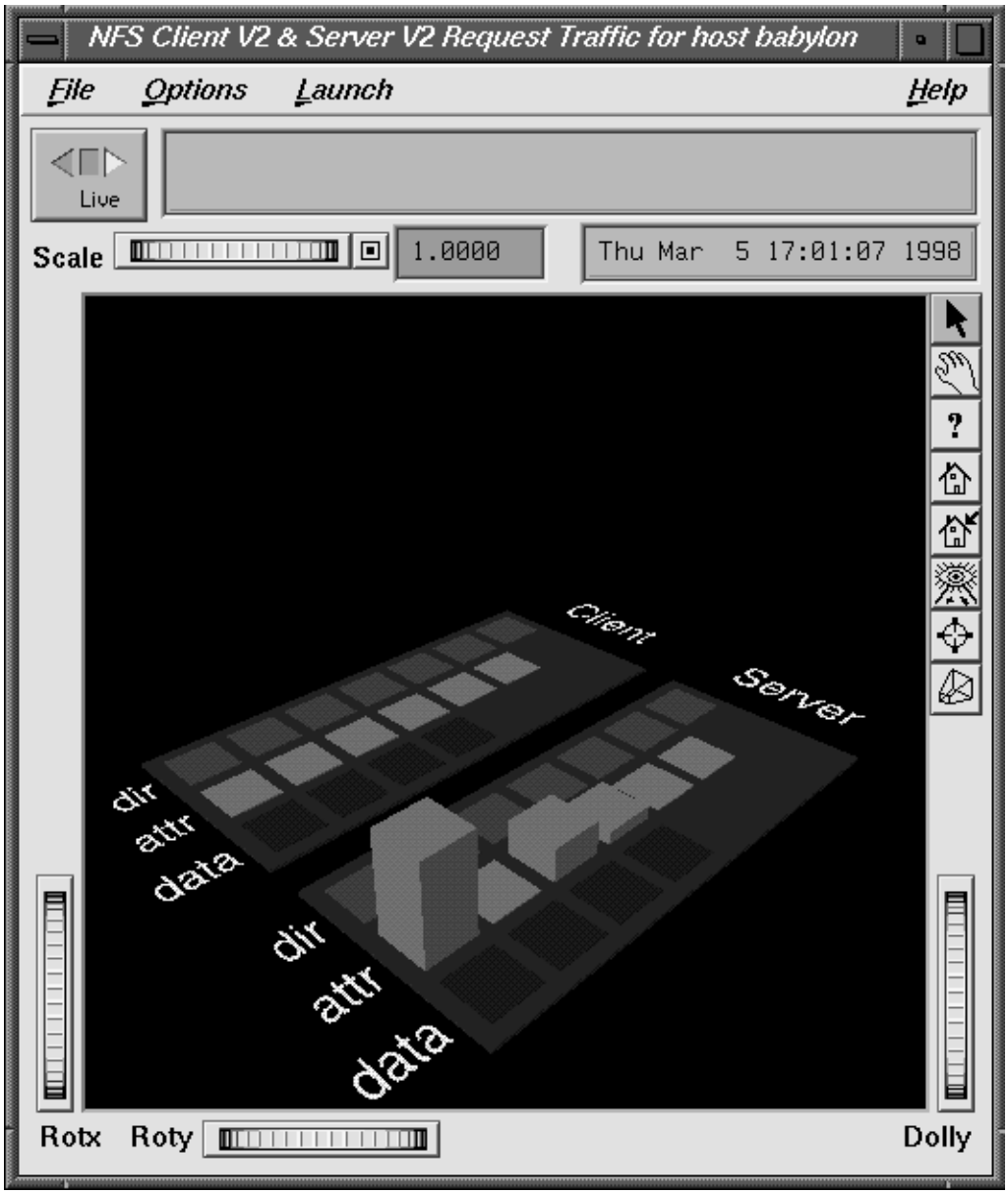


**Figure 5-4** nfsvis Window

## The oview Origin Visualization Tool

The *oview* tool displays a dynamic display of Origin system topology and performance, as shown in Figure 5-5. It displays performance information about CPUs, nodes, and routers in Origin systems connected in various configurations; see oview(1) for details.

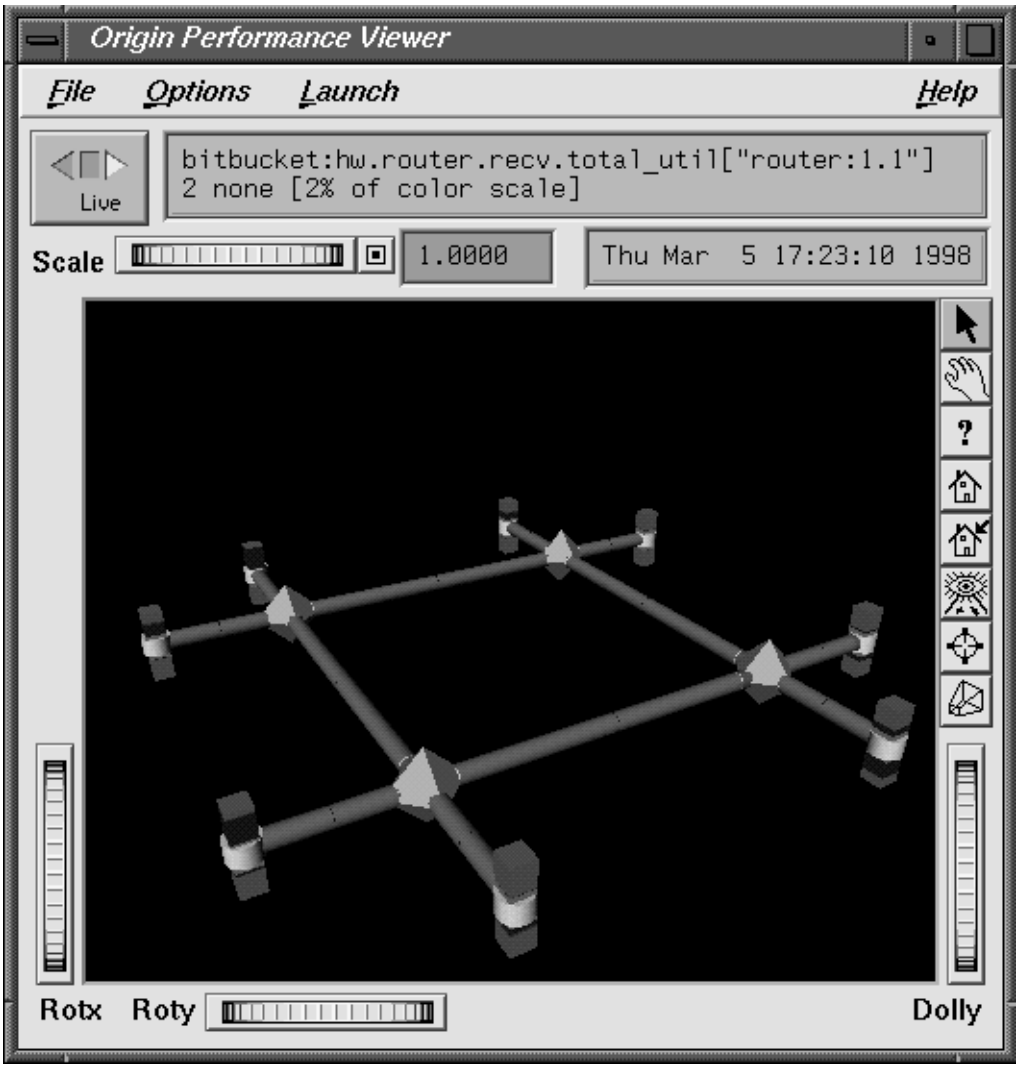


**Figure 5-5** oview Window

## The pmview Tool

The *pmview* tool is a generalized three-dimensional 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 5-1, Figure 5-2, Figure 5-3, and Figure 5-4.

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 ivview(1), which displays the 3D scene.

The left, right, and bottom edges of the examiner viewer window contain a variety of thumbwheels and buttons that allow the user to adjust the visualization of the 3D scene. The *Rotx* and *Roty* thumbwheels allow the user to rotate the scene about the X and Y axes, respectively. The *dolly* thumbwheel 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. This is described in detail below. The Escape 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 "dolly" the scene 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 5-6. 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.

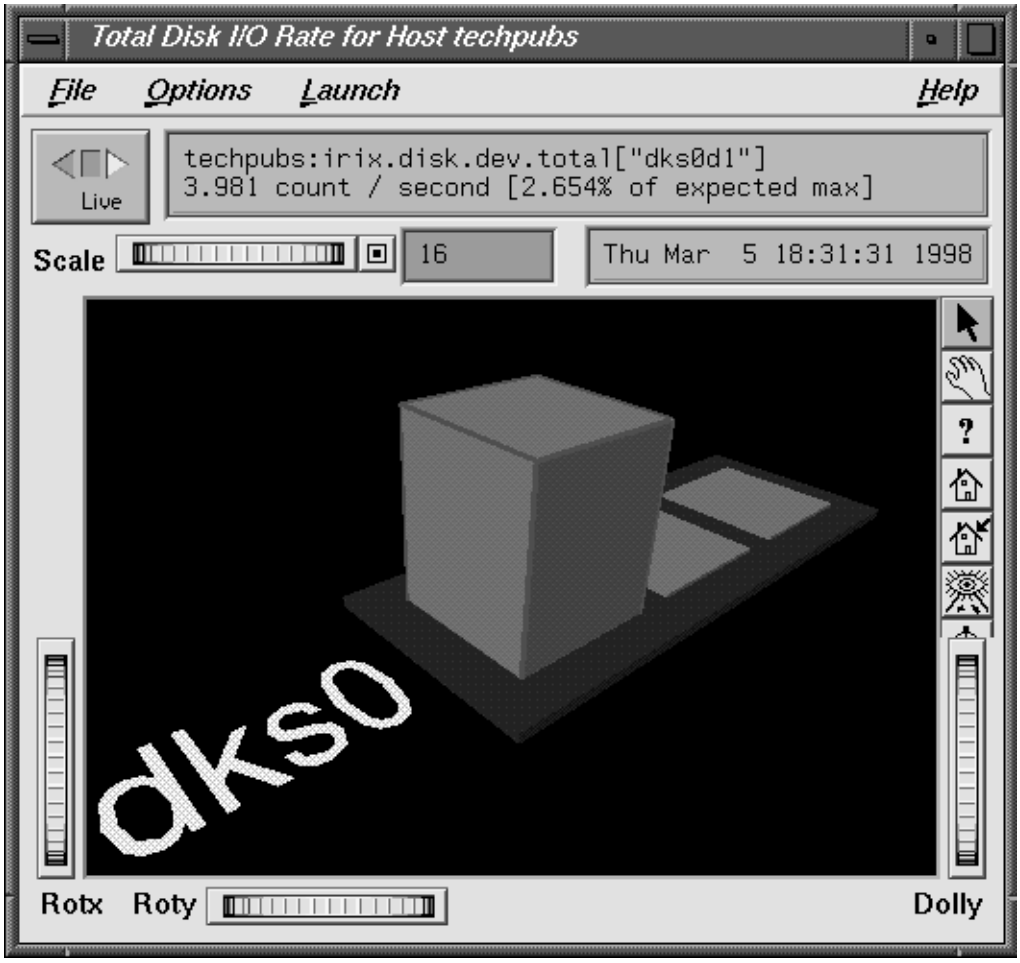


**Figure 5-6** pmview Window With a Block Selected

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.

There are four menus in *pmview* tools: the File menu for recording, saving, and printing scenes, the Options menu for accessing the time controls, the Launch menu for starting other tools, and the Help menu for obtaining 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, then 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 *nodevis* is launched, then 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.

*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 “The pmchart Tool” on page 54 for details.

*pmgirix*
: 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 “The pmval Command” on page 75 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 "Time Duration and Control" on page 41 for a complete description of the time control services). Pressing this button displays the time control dialog.

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

The button beside the thumbwheel 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.

## Creating Custom Visualization Tools With pmview

At startup time, a configuration file is read that specifies

- the 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 this:

```
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 (
                irix.kernel.percpu.cpu.idle[cpu0]        1000 "idle"
                irix.kernel.percpu.cpu.wait.total[cpu0]  1000 "wait"
                irix.kernel.percpu.cpu.intr[cpu0]        1000 "intr"
                irix.kernel.percpu.cpu.sys[cpu0]         1000 "sys"
                irix.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).

As an example, the following specification file produces a scene like the one shown in Figure 5-7. The file has a grid, labels, bars and a stack utilization object.

```
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 (
            irix.kernel.all.cpu.user        1000
            irix.kernel.all.cpu.sys         1000
            irix.kernel.all.cpu.intr        1000
            irix.kernel.all.cpu.wait.total  1000
            irix.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 (
            irix.kernel.all.load[15]    2
            irix.kernel.all.load[5]     2
            irix.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 (
            irix.mem.util.kernel        1
            irix.mem.util.fs_ctl        1
            irix.mem.util.fs_dirty      1
            irix.mem.util.fs_clean      1
            irix.mem.util.user          1
        )
        _colorlist memory_colors
        _baseLabel "Physical Memory Utilization"
    )
    _label 0 3 _down _large "Disk"
    _stack 1 3 _west _cylinder (
        _metrics (
            irix.disk.all.read          100
            irix.disk.all.write         100
        )
        _colorlist disk_colors
        _baseLabel "Disk Operations\nNormalized to 100 I/Os per second"
    )
)
```

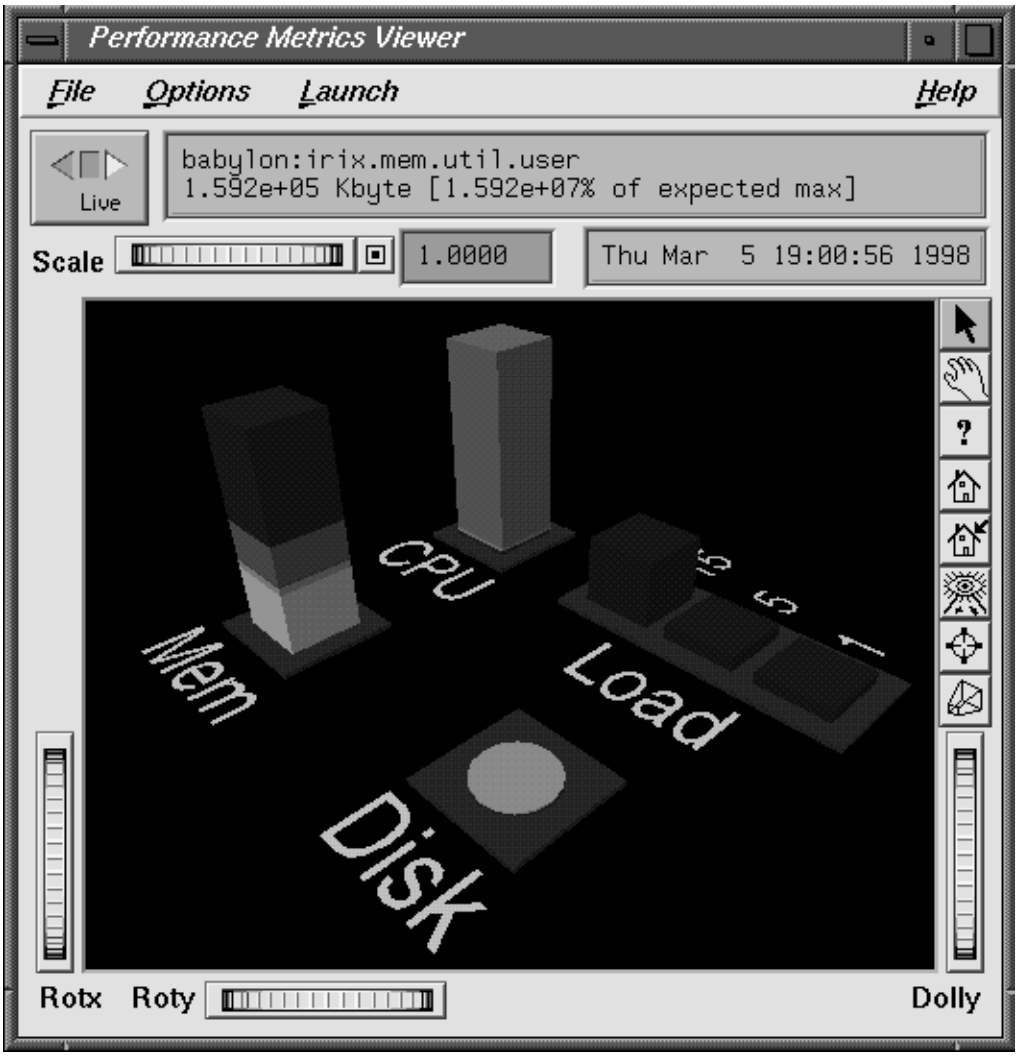


**Figure 5-7** Custom pmview Specification

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*.

*Chapter 6*

# Performance Metrics Inference Engine

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

The following major sections in this chapter are as follows:

- "Introduction to pmie" on page 104 provides an introduction to the concepts and design of *pmie*.
- "Basic pmie Usage" on page 106 describes the basic syntax and usage of *pmie*.
- "Specification Language for pmie" on page 111 discusses the complete *pmie* rule specification language.
- "Some Real pmie Examples" on page 126 provides an illustration by example, covering several common performance scenarios.
- "Developing and Debugging pmie Rules" on page 129 presents some tips and techniques for *pmie* rule development.
- "Caveats and Notes on pmie" on page 129 presents some important information on using *pmie*.
- "Creating pmie Rules With pmrules" on page 131 provides a brief description of how to use the *pmrules* GUI for creating *pmie* rules from parameterized templates.

## Introduction to pmie

Automated reasoning within Performance Co-Pilot 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 Subsystem (PMCS) and delivered through the Performance Metrics Application Programming Interface (PMAPI).

Typical *pmie* uses would 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 to guide performance analysis, trend analysis, and capacity planning.

The *pmie* "expressions" are described in a language that supports a wide range of expressive power and operational flexibility, and 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. Pre-defined actions include
  - launch a visible alarm with *xconfirm*; see xconfirm(1)
  - post an entry to the system log */var/adm/SYSLOG*; see syslog(3C)
  - 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; most 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 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 sub-expressions 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 looming, 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 *pmrules* tool provides a convenient customization method, allowing the user to generate parameterized *pmie* rules for some of the more common performance scenarios.

## 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, "Common Conventions and Arguments." In addition, *pmie* accepts the following command-line arguments:

| | |
|---|---|
| **-d** | Enable 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.

## 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 "Conceptual Foundations" on page 11. 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 *pminfo* 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 irix.disk.dev.total
```

This produces the following response:

```
irix.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 `irix.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 irix.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* **irix.mem.freemem irix.kernel.percpu.syscall**

In response, you see output similar to this:

```
irix.mem.freemem
    Data Type: 32-bit unsigned int  InDom: PM_INDOM_NULL 0xffffffff
    Semantics: instant  Units: Kbyte

irix.kernel.percpu.syscall
    Data Type: 32-bit unsigned int  InDom: 1.1 0x400001
    Semantics: counter  Units: count
```

**Note:** A cumulative counter such as `irix.kernel.percpu.syscall` is automatically converted by *pmie* into a rate (measured in events per second, or count/second), while instantaneous values such as `irix.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 `irix.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).

## Simple pmie Example

The following 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 `irix.disk.dev.total`, collected from the local *pmcd*:

```
pmie -v
iops = irix.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, and 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 entered on standard input (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) reference page for a complete description of *pmie* command line options.

## Complex pmie Examples

This section illustrates more complex *pmie* expressions with a view to establishing the flavor of the specification language. The next section provides a complete description of the *pmie* specification language.

The arithmetic expression

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

computes the percentage of write operations over the total number of disk transfers. The `irix.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, `irix.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:

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

The value is true if the number of writes exceeds the number of reads, and if there is some reasonably 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 irix.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 = (irix.disk.all.write / irix.disk.all.total) * 100;
busy_wrt = irix.disk.dev.total > 10 &&
           irix.disk.dev.write > irix.disk.dev.read;
busy = some_inst irix.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).

## 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 "Some Real pmie Examples" on page 126.

Complex expressions are built up recursively from simple elements:

1. Performance metric values are obtained from *pmcd* for real-time or live sources, otherwise from PCP archive logs.
2. Metric 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 very 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.

## Basic pmie Syntax

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

### Lexical Elements

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

identifier
: Begins with an alphabetic (either upper or lowercase), followed by zero or more letters chosen from the alphabetics, the numeric digits, and the special characters period (`.`) and underscore (_); for example, `x`, `irix.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 the 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 constants
: 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; for example, `"A \"gentle\" reminder"`.

### Comments

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

`/*` *text* `*/`
: C-style comment, optionally spanning multiple lines, with no nesting of comments.

`//` *text*
: C++-style comment from here to the end of the line.

### Macros

When 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 form

*identifier = "string";*

associates the macro name *identifier* with the given *string* constant.

Any subsequent occurrence of

**$***identifier*

is replaced by the *string* most recently associated with a macro definition for *identifier*. For example, given the macro definition

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

you can then use the 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.

### 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 white 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**, **Tcount**. Plural forms are also accepted.

The following are examples of units usage:

```
irix.disk.dev.blktotal > 1 Mbyte / second;
irix.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 expression `irix.mem.freemem < 500`, the `500` is interpreted as 500 bytes.

## 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.

## pmie Metric Expressions

A Performance Metrics Name Space (PMNS) provides a means of naming performance metrics, for example, `irix.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 metric 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 the Internet naming conventions.

The names for instances are provided by the Performance Metrics Domain Agents (PDMA) 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, stretching back 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 6-1.

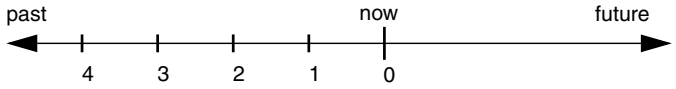


**Figure 6-1** Sampling Time Line

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 "Setting Evaluation Frequency" on page 114.

For *pmie*, a metric expression is the name of a metric, optionally qualified by a host, instance and sample time specification. Special characters introduce the qualifiers, namely colon (`:`) for hosts, hash or pound sign (`#`) for instances, and at (`@`) for sample times. For example, the expression

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

refers to the previous value (`@1`) of the counter for the disk read operations associated with the disk instance `#dks0d1` on the host `moomba`. In fact, this expression defines a point in the three-dimensional parameter space of {*host*} x {*instance*} x {*sample time*} as shown in Figure 6-2.

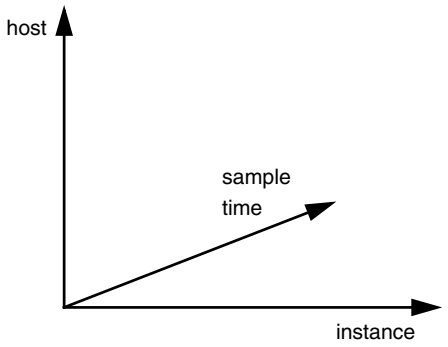


**Figure 6-2** Three-Dimensional Parameter Space

A metric expression may also identify sets of values corresponding to one-, two-, or three-dimensional 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 "pmie and the Performance Metrics Collection System" on page 107.

   Within the *pmie* grammar, an instance name is an identifier, so if the instance name contains characters other than the alphanumerics, it is probably safest to 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 encoded as 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.

Putting all of this together, the metric expression

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

refers to a three-dimensional 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.

## pmie Rate Conversion

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

```
irix.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:

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

However, as well as being too verbose, the accuracy of *delta* as a measure of actual inter-sample delay is an issue here. *pmie* requests samples, at intervals of approximately *delta*; the results exported from the PMCS are time stamped with the high-resolution system clock at the time 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. That is, the expression

```
irix.disk.all.total
```

is unconditionally converted to a rate by *pmie*.

## pmie Arithmetic Expressions

Within *pmie* simple arithmetic expressions are constructed from metric expressions (see "pmie Metric Expressions" on page 114) 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.

The section "pmie Intrinsic Operators" on page 125 describes additional operators that may be used for aggregate operations to reduce the dimensionality of an arithmetic expression.

## pmie Logical Expressions

A number of logical expression types are supported, described in the following sections:

- logical constants
- relational expressions
- boolean expressions
- quantification operators

### Logical Constants

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

### 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,

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

is a relational expression that is true or false, depending on the aggregate total of disk read operations per second being greater than 50.

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

As described in "pmie Metric Expressions" on page 114, arithmetic expressions in pmie may assume set values. Hence the relational operators are also required to take as arguments constant, singleton, and set-valued expressions; the result has the same dimensionality as the operands. So, given the rule

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

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 ">" produces six truth values for the cross-product of the *instance* and *sample time* dimensions.

The section "Quantification Operators" on page 120 describes additional logical operators that may be used to reduce the dimensionality of a relational expression.

### Boolean Expressions

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

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

### 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 the section "pmie Metric Expressions" on page 114. The *pmie* quantification operators perform this function.

Each quantification operator takes a one-, two-, or three-dimensioned set of truth values as an operand, and reduces it to a set of smaller dimension, by quantification along a single dimension. For example, if the expression in the previous example is simplified and prefixed by `some_sample`, to produce

```
intfs = "#ec0 #ec2";
all_intf = some_sample irix.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 expression

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

answers the question: "Are all hosts experiencing at least 20% of their disks busy either reading or writing?"

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

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

**Note:** To avoid confusion over precedence and scope for the quantification operators, the use of explicit parenthesis is encouraged.

## 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; that is, aggregation and quantification operators 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 "Setting Evaluation Frequency" on page 114. 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, using *logger*. An optional first *argument* is interpreted as a *priority* (see the **-p** option for *logger*); 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.

Clearly, some ambiguity may occur in respect of which *host*, *instance,* or a performance metric is bound to a %-token. In most cases, the leftmost binding in the top-level subexpression is used, and this accommodates the vast majority of common rules. 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 subexpression contribute to the %-token bindings.

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

```
some_inst ( irix.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 `irix.disk.dev.total` that make the expression true. If more than one instance makes the expression true (that is, 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

```
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  = "irix.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
```

## pmie Intrinsic Operators

The following sections describe some further useful intrinsic operators for *pmie*:

- arithmetic aggregation
- the `rate` operator
- transitional operators

### 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` Compute the average value across all *instances* for the same *host* and *sample time*.

`sum_host` Compute the total value across all *instances* for the same *host* and *sample time*.

`count_host` Compute the number of values across all *instances* for the same *host* and *sample time*.

`min_host` Determine the minimum value across all *instances* for the same *host* and *sample time*.

`max_host` Determine the maximum value across all *instances* for the same *host* and *sample time*.

And there are ten further operators corresponding to the forms `*_inst` and `*_sample`.

The following example illustrates the use of an aggregate operator in combination with a 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 ( irix.kernel.percpu.cpu.user $poke +
                 irix.kernel.percpu.cpu.sys $poke > 0.7 ) >= 2
    )
        -> alarm "2 or more busy CPUs";
```

### The rate Operator

The `rate` operator computes the rate of change of an arithmetic expression. For example:

```
rate irix.mem.freemem
```

returns the rate of change for the `irix.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"pmie Rate Conversion" on page 117) and the `rate` operator would produce the second derivative with respect to time, which is less likely to be useful.

### 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.

## Some Real pmie Examples

The examples presented in this section are task-oriented and use the full power of the *pmie* specification language as described in "Specification Language for pmie" on page 111.

Source code for the *pmie* examples in this chapter, and many more, is provided in the PCP subsystem *pcp.sw.demo*, and when installed may be found in */var/pcp/demos/pmie*. The following examples are given here:

- monitoring CPU utilization
- monitoring disk activity

### Monitoring CPU Utilization

```
// Some Common Performance Monitoring Scenarios
//
// The CPU Group
//

delta = 2 sec;  // more often for demonstration purposes

// common prefixes
//
percpu  = "irix.kernel.percpu";
all     = "irix.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 ";
```

### Monitoring Disk Activity

```
// Some Common Performance Monitoring Scenarios
//
// The Disk Group
//

delta = 15 sec;         // often enough for disks?

// common prefixes
//
disk    = "irix.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 ";
```

## 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 like this:

```
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.

## Caveats and Notes on pmie

The following sections provide important information for users of *pmie*.

### Performance Metric Wrap-Around

Performance metrics that are cumulative counters may occasionally overflow their range and wrap around 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.

### 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 as fast as hundreds of times per second, using small sample intervals creates unnecessary load on the monitored system.

## 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 white-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 to 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.

## pmie Error Detection

The parser used in *pmie* is currently not robust in the face of syntax errors. It is suggested that you check any problem 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.

## 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.

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 6-3.

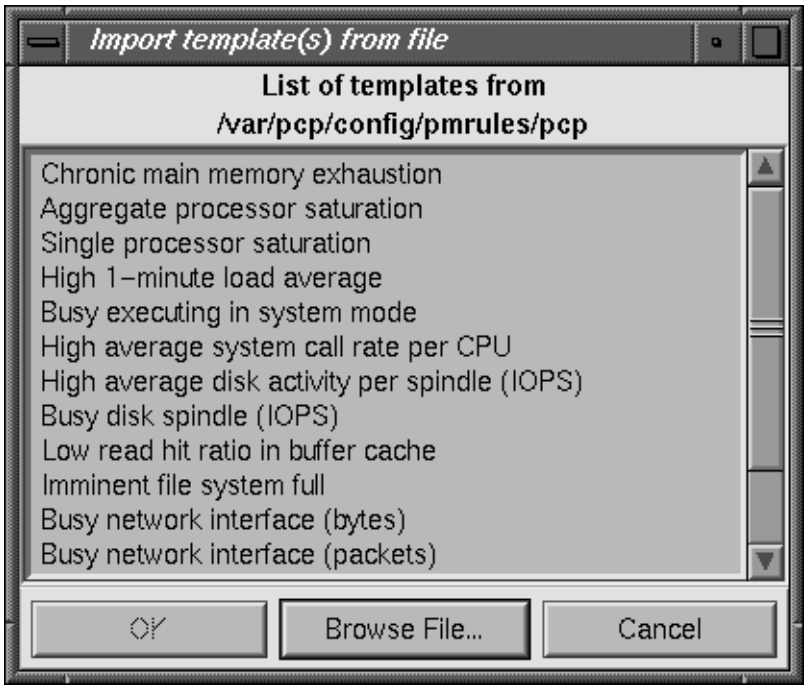


**Figure 6-3** pmrules Import Template Dialog

4. Select the desired templates, click OK, and return to the *pmrules* main window, which appears similar to the one shown in Figure 6-4.

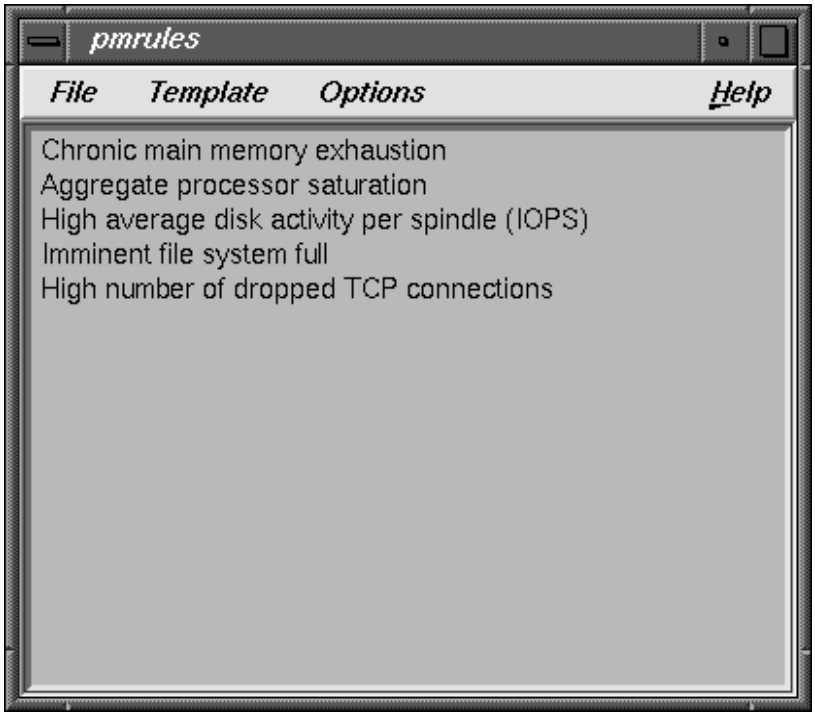


**Figure 6-4** pmrules Main Dialog After Template Selection

5. Double-click the desired template, and the *pmrules* Edit template dialog displays, similar to the one shown in Figure 6-5.

   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.

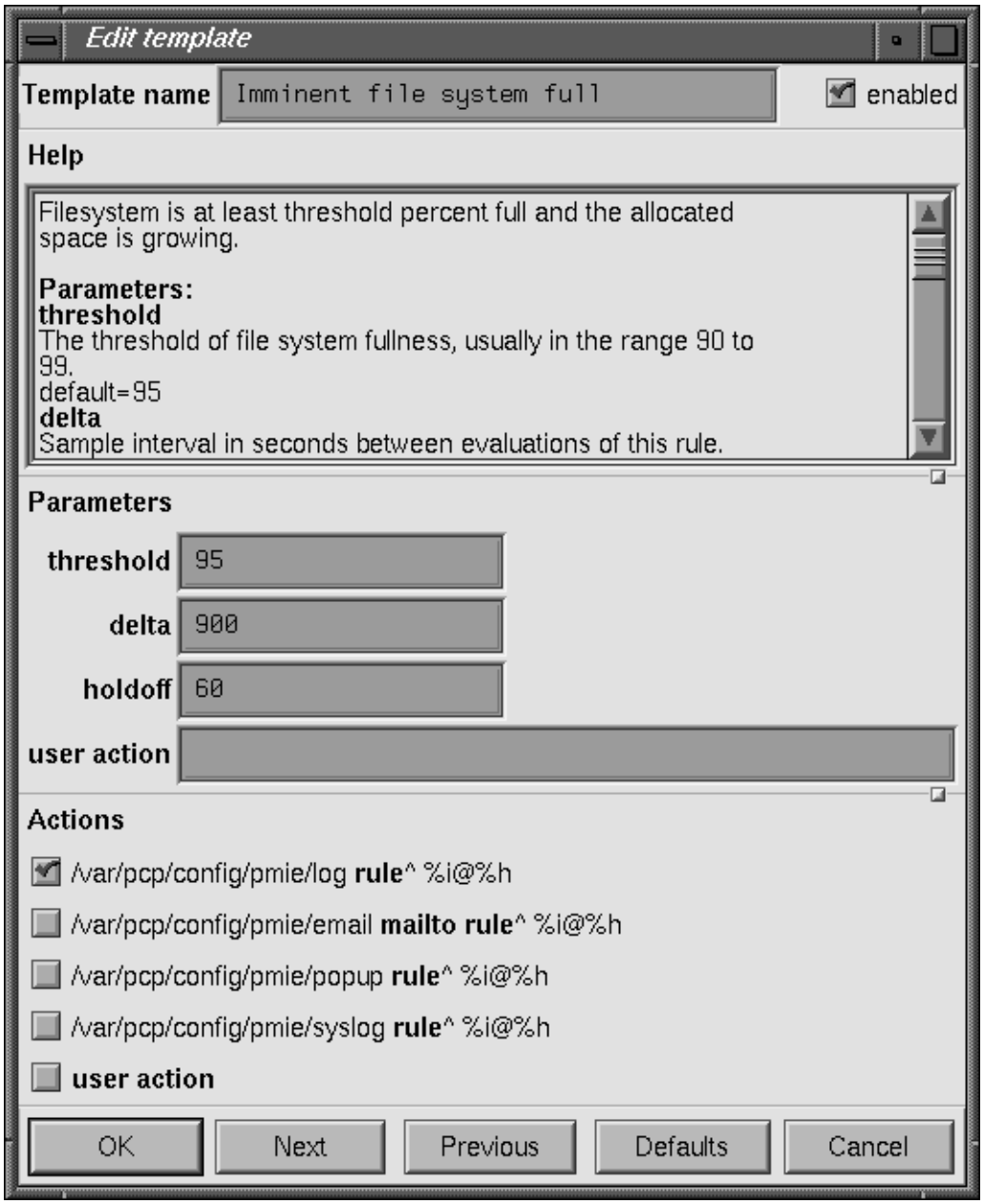


**Figure 6-5** pmrules Edit Template Dialog

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) reference page for a complete description of the capabilities of the *pmrules* tool.

*Chapter 7*

# Archive Logging

Performance monitoring and management in complex systems demands the ability to accurately capture performance characteristics for subsequent review, analysis, and comparison. Performance Co-Pilot 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:

- "Introduction to Archive Logging" on page 135 presents the concepts and issues involved with creating and using archive logs.
- "Using Archive Logs and Performance Visualization Tools" on page 137 describes the interaction of the PCP tools with archive logs.
- "Cookbook for Archive Logging" on page 142 shows some shortcuts for setting up useful PCP archive logs.
- "Archive Logging Troubleshooting" on page 150 presents helpful direction if your archive logging implementation is not functioning correctly.

## 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 *oview*, *pmchart*, and *pmview* 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 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.

#### Archive Logs and the PMAPI

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

Live feeds 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 current or historical performance data.

#### 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 post-mortem. 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 on 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 *cron*) to detect performance regressions.

For more about *pmie*, see Chapter 6, "Performance Metrics Inference Engine."

#### 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 *cron.snap* services may be used to automate snapshots.

#### 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.

## Using Archive Logs and 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.

## Coordination Between pmlogger and PCP tools

Most commonly, a PCP tool would be invoked with a **-a** option to process an archive log some time after *pmlogger* had finished creating the archive. However, a tool such as *pmchart* that uses a Time Control dialog (see the section "Time Duration and Control" on page 41) 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 the end of the archive may be aligned with an I/O buffer boundary, rather than with a logical archive log record, and this might confuse the PCP tool. These problems may be avoided by sending *pmlogger* a SIGUSR1 signal, or using the `flush` command of *pmlc* to force *pmlogger* to flush its output buffers.

## Administering PCP Archive Logs Using cron Scripts

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

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

*cron.pmdaily* This script performs a daily "housecleaning" of archive logs and notices.

*cron.pmlogmerge* This script is used to merge archive logs and is called by *cron.pmdaily*.

*cron.pmcheck* This script checks to see that all desired *pmlogger* processes are running on your system, and invokes any that are missing for any reason.

*cron.pmsnap* This script generates graphic image snapshots of *pmchart* performance charts at regular intervals.

These scripts can be edited to suit your particular needs, and are generally controlled by the */var/pcp/config/pmlogger/control* file (*cron.pmsnap* has an additional control file). Complete information on these scripts is available in the cron.pmdaily(1) and cron.pmsnap(1) reference pages.

## 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 upcoming sections provide information to assist you in PCP archive log file management.

### 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 7-1.

**Table 7-1** 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,... |

### Log Volumes

A single PCP archive may be partitioned into a number of volumes. These volumes may expedite management of the archive; however, all volumes must typically 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 takes a numeric argument specifying the number of performance metric samples to be written to each log volume 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.irix.log.0
netserver.irix.log.1
netserver.irix.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) reference page.

### 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 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 YYMMDD.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 YYMMDD. The *cron.pmdaily* script performs this action and a number of other routine housekeeping chores.

### 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 "PCP Archive Logger Deployment" on page 159. 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 7-1.

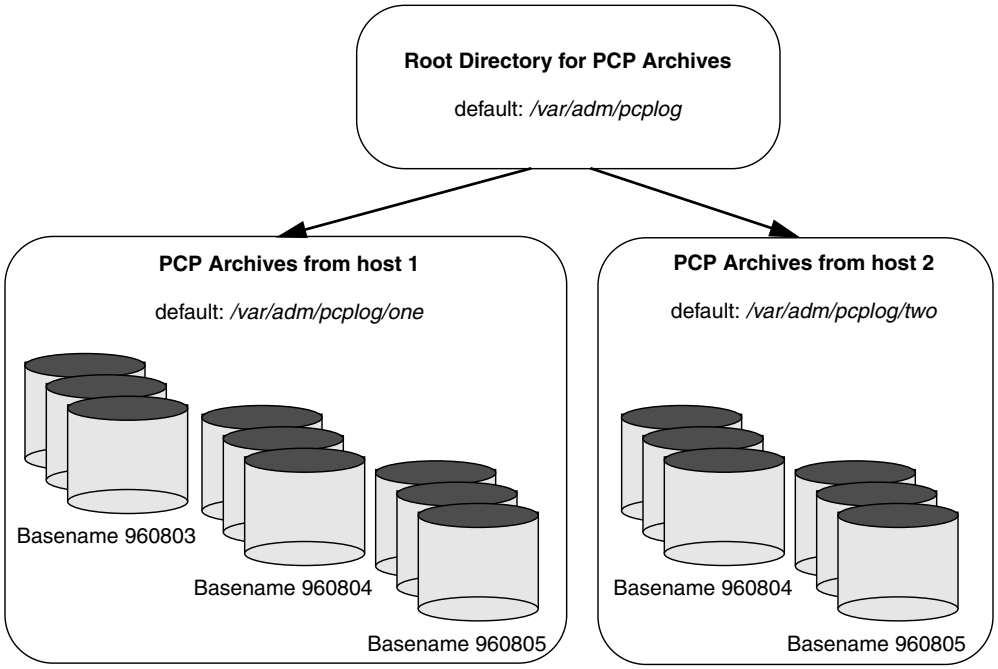


**Figure 7-1** Archive Log Directory Structure

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.

### 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 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 are mandatory *maybe* and advisory *off* by default, so are not logged.

A complete description of the *pmlogger* configuration format can be found on the pmlogger(1) reference page.

#### 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, the command

```
pmdumplog -l /var/adm/pcplog/www.sgi.com/960731
```

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 irix.network.mbuf
irix.network.mbuf.alloc
irix.network.mbuf.typealloc
irix.network.mbuf.clustalloc
irix.network.mbuf.clustfree
irix.network.mbuf.failed
irix.network.mbuf.waited
irix.network.mbuf.drained
```

**Note:** In PCP version 2.0 archives, the collector's PMNS is stored in the meta file.

## Cookbook for Archive Logging

This section presents 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 reference pages for *pmlogger* and related tools.

## 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 -1 /var/adm/pcplog/pluto
960808.14.33.0
960808.14.33.index
960808.14.33.meta
Latest
pmlogger.log
```

## 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/cron.pmcheck
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
```

## 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 *cron.pmdaily* and *cron.pmcheck*; for example:

   ```
   # daily processing of archive logs
   10     0     *     *     *       /usr/pcp/bin/cron.pmdaily
   # every 30 minutes, check pmlogger instances are running
   25,55  *     *     *     *       /usr/pcp/bin/cron.pmcheck
   ```

3. Make these changes permanent with this command:

   ```
   crontab </tmp/foo
   ```

## 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 *cron.pmsnap*; instructions for this operation can be found in "Taking Snapshots of pmchart Displays and Value Dialogs" on page 68.

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_gifts.sw.pmsnap* 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/cron.pmcheck`**
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 *cron.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 *cron.pmsnap*; for example:

   ```
   # every 30 minutes, generate performance snapshot images
   30,0   *     *     *     *       /usr/pcp/bin/cron.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 a 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 xhost(1) for details. If this host is `grover`, then replace

   ```
   -d :0
   ```

   in the line above with

   ```
   -d grover:0
   ```
7. Make these changes permanent with this command:

   **`crontab </tmp/foo`**
8. After 30 minutes or so (time enough for *cron* 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*.

## 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 PCP clients like *pmchart* and *pmview.*

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 7-1, 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) reference 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.

## Manipulating Archive Logs With pmlogextract

The *pmlogextract* tool takes a number of PCP archive logs from a single host and

- merges the archives into a single log, while maintaining the correct timestamps 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) reference page for full information on this command, which replaces functionality of the *pmlogmerge* tool as of PCP release 2.0.

## 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 root 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 started in this fashion, the file */etc/config/pmlogger.options* 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.

## Using pmlc

You may tailor *pmlogger* dynamically with the *pmlc* command. Normally the *pmlogger* configuration is read at startup, and 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) reference page for a description of command-line options. When 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 irix.disk.dev.read
pmlc> query irix.disk.dev
irix.disk.dev.read
    adv  on  nl       5 min  [131073 or “dks0d1”]
    adv  on  nl       5 min  [131074 or “dks0d2”]
pmlc> quit
```

Refer to the pmlc(1) and pmlogger(1) reference pages for complete details.

## Archive Logging Troubleshooting

The following issues concern the creation and use of logs using *pmlogger*.

### 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 command, then re-run *pmlogger*:

```
rm -f foo.*
```

### 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 routine `_pmOpenLog`. For 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.

### 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 aren't 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.

## 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 *ps*.

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
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*.

## 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 with pmlogcheck(1).

*Chapter 8*

# Performance Co-Pilot Deployment Strategies

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:

- "Basic Deployment" on page 154 presents the spectrum of deployment architectures at the highest level.
- "PCP Collector Deployment" on page 157 describes alternative deployments for PMCD and the PMDAs.
- "PCP Archive Logger Deployment" on page 159 covers alternative deployments for the *pmlogger* tool.
- "PCP Inference Engine Deployment" on page 162 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.

## Basic Deployment

In the simplest PCP deployment, one system is configured as both a collector and a monitor, as shown in Figure 8-1. Because the PCP monitor tools make extensive use of visualization, this suggests the single system would be configured with a graphics head.

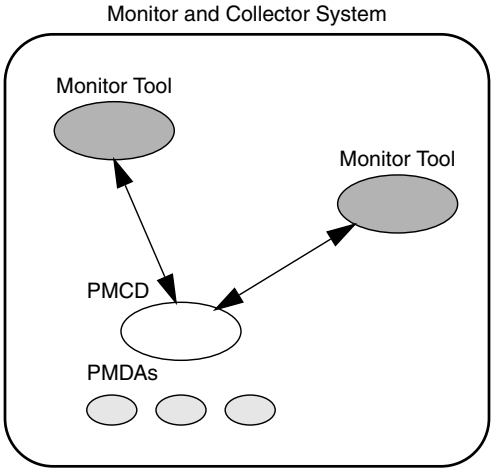


**Figure 8-1** PCP Deployment for a Single System

However, most PCP deployments involve at least two systems. For example, the setup shown in Figure 8-2 would be representative of many common scenarios.

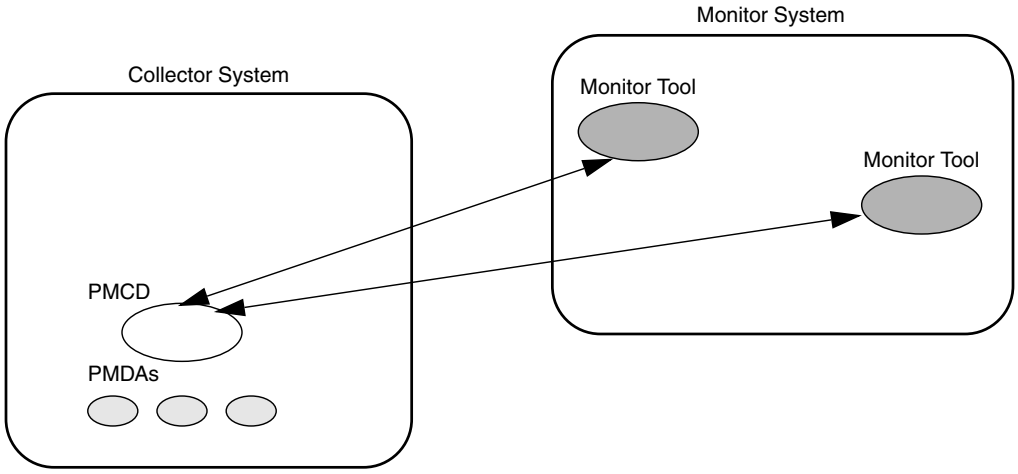


**Figure 8-2** Basic PCP Deployment for Two Systems

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 8-3.

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 8-3 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.

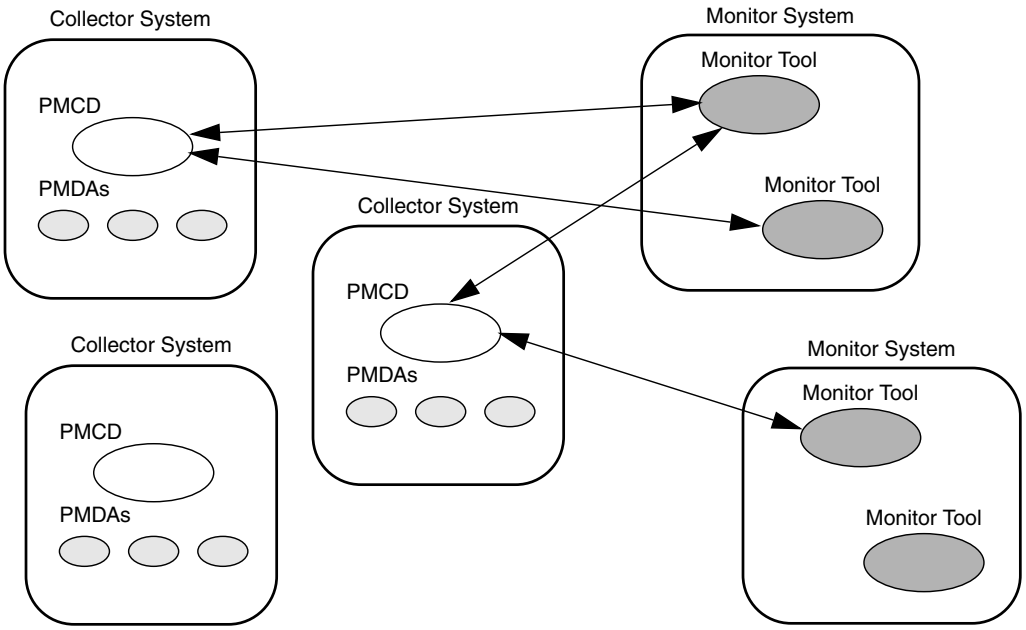


**Figure 8-3** General PCP Deployment for Multiple Systems

## PCP Collector Deployment

Each PCP collector system must have an active *pmcd* (which implies a valid PCP Collector license), and typically a number of PMDAs installed.

### Principal Server Deployment

Those hosts first selected as PCP collector systems are likely to be the ones that provide some class of service deemed to be critical to the information processing activities of the enterprise. Examples include

- 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*.

## 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 8-4 illustrates one way of measuring quality of service for client-server applications.

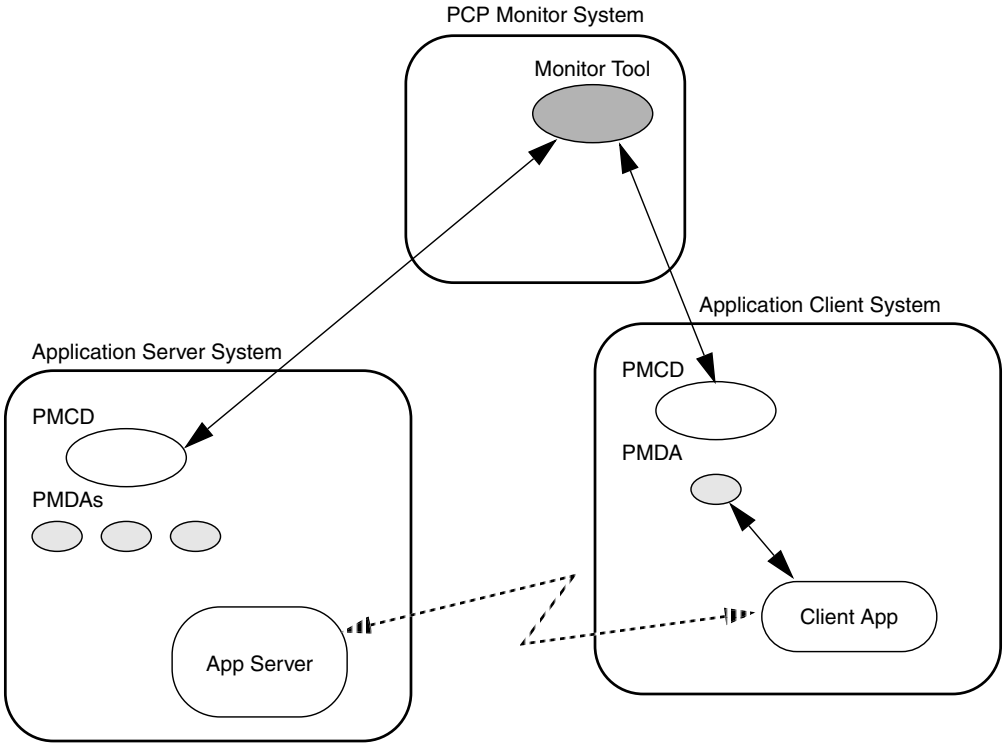


**Figure 8-4** PCP Deployment to Measure Client-Server Quality of Service

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 concurrently monitored.

PCP add-on products implement a number of examples of this architecture, including the *webping* PMDA for Web servers, the *oraping* PMDA for an Oracle DBMS, the *sybping* PMDA for a Sybase DBMS, and the *infmxping* PMDA for an INFORMIX DBMS.

For each of these PMDAs, the full source code is distributed with the associated add-on 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.

## PCP Archive Logger Deployment

PCP archive logs are created by the *pmlogger* utility, as discussed in Chapter 7, "Archive Logging," and 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.

## Deployment Options

The issue is relatively simple and reduces to "On which host(s) should *pmlogger* be running?" The options are these:

1. Run *pmlogger* on each PCP collector system to capture local performance data.
2. Run *pmlogger* on some of the PCP monitor systems to capture performance data from remote PCP collector systems.
3. 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 8-5.

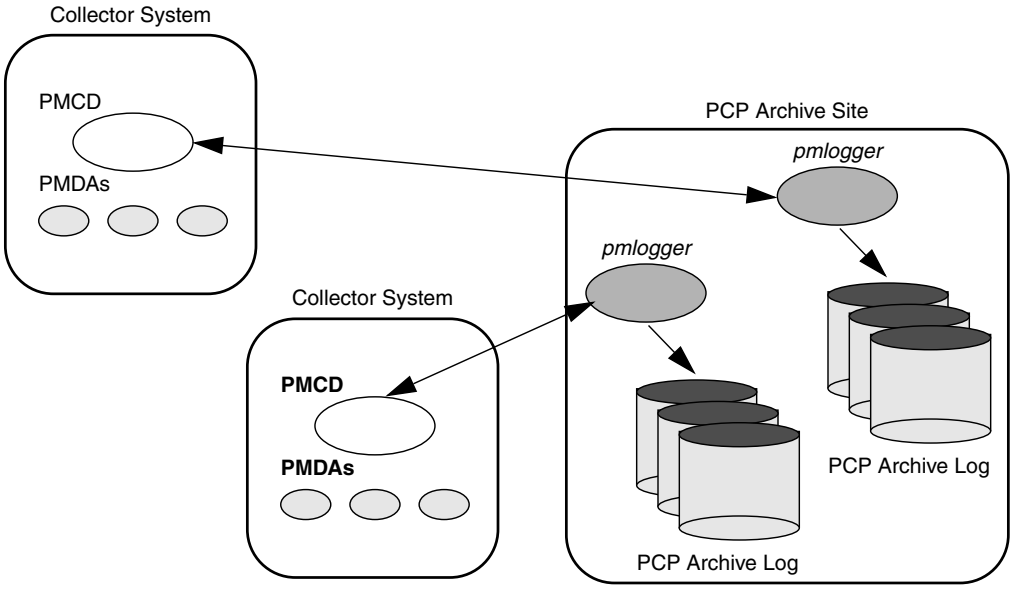


**Figure 8-5** Designated PCP Archive Site

## 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 KB and 10 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.

## Operational Management

There is an initial administrative cost associated with configuring each *pmlogger* instance, and an on-going administrative investment to monitor these configurations, perform regular house-keeping (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 “Archive Log File Management” on page 139. However, the necessity and importance of these tasks favor a centralized *pmlogger* deployment, as shown in Figure 8-5.

**Note:** The *pmlogger* utility is not subject to any PCP license restrictions, and may be installed and used on any host.

## Exporting PCP Archive Logs

Collecting PCP archive logs is of little value unless the logs are processed as part of the on-going 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 forwards and backwards 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 "Archive Log File Management" on page 139 for details). You must have concurrent access to all of these files before a PCP tool is able to process an archive log correctly.

## PCP Inference Engine Deployment

The *pmie* utility supports automated reasoning about system performance, as discussed in Chapter 6, "Performance Metrics Inference Engine," 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.

### 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:

1. For real-time analysis, run *pmie* on each PCP collector system to monitor local system performance.
2. For real-time analysis, run *pmie* on some of the PCP monitor systems to monitor the performance of remote PCP collector systems.
3. For retrospective analysis, run *pmie* on the systems where the PCP archive logs reside. The problem then reduces to *pmlogger* deployment as discussed in "PCP Archive Logger Deployment" on page 159.
4. 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 8-6.

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 8-6.

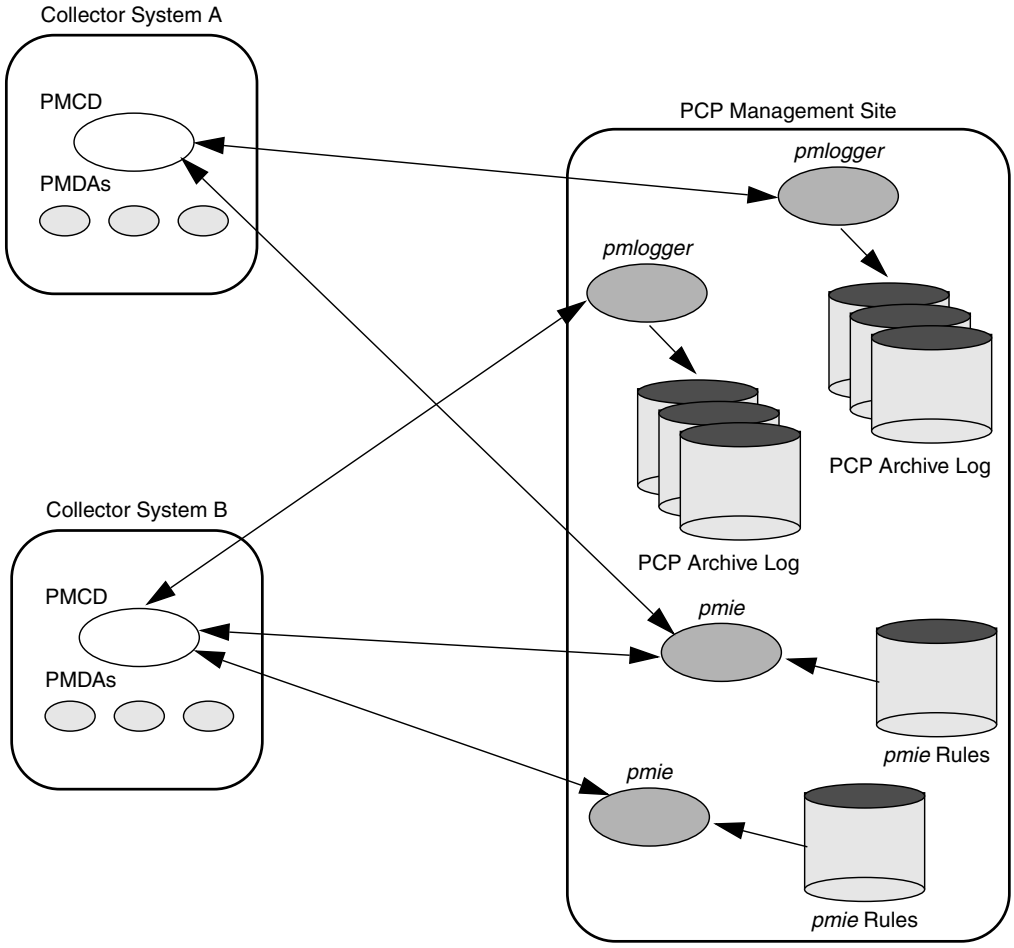


**Figure 8-6** PCP Management Site Deployment

## 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.

## 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. 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 8-6.

However, it is most likely that knowledgable 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.

**Note:** The *pmie* utility is subject to PCP Monitor license restrictions, and hence uncontrolled distribution is unlikely.

*Chapter 9*

# Customizing and Extending PCP Services

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:

- "PMDA Customization" on page 165 describes the general process of installing and removing a PMDA at both a PCP collector and/or a PCP monitor host.
- "Customizing the Summary PMDA" on page 166 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.
- "PCP Tool Customization" on page 169 describes the various options available for customizing and extending the basic PCP tools.
- "PMNS Management" on page 173 covers the concepts and tools provided for updating the PMNS (Performance Metrics Name Space).
- "PMDA Development" on page 176 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.
- "PCP Tool Development" on page 176 outlines how new tools may be developed to process performance data from the PCP infrastructure.

## PMDA Customization

The generic procedures for installing and activating the optional PMDAs has been described in "Managing Optional PMDAs" on page 30. 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.

## 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) reference page.

The summary PMDA consists of two processes:

*pmie* process — A dedicated instance of the PCP *pmie* inference engine is used to periodically sample the base metrics and compute values for the derived metrics. This *pmie* process is launched with special command-line arguments by the main process. See the "Introduction to pmie" on page 104 for a complete discussion of the *pmie* feature set.

main process — The main process reads and buffers the values computed by the *pmie* process and makes them available to the Performance Metrics Collector 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 the section "PMDA Installation" on page 30.

Alternatively, the summary PMDA may be customized to export your own derived performance metrics, by following the procedure described below:

1. Choose Performance Metric Name Space (PMNS) names for the new metrics. These must begin with `summary` and follow the rules described in pmns(4). For example, you might use `summary.fs.cache_write` and `summary.fs.cache_hit`.
2. Edit the file *pmns* in the */usr/pcp/pmdas/summary* directory to add the new metric names in the format described in pmns(4). 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 "PMNS Management" on page 173 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
}
```

3. 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
   ```

4. Edit the file */usr/pcp/pmdas/summary/expr.pmie* 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 "Specification Language for pmie" on page 111.

   For example, consider this expression:

   ```
   // filesystem buffer cache hit percentages
   prefix = "irix.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;
   ```

5. 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
```

6. 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
   ```

7. 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.

8. You may now create *pmchart* views, *pmview* scenes, and *pmlogger* configurations to monitor and archive your new performance metrics.

## PCP Tool Customization

The 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 reference pages, and in the files that are made available as part of the PCP installation.

### Stripchart Customization

The PCP tool *pmchart* produces stripchart displays of performance metrics. Refer to "The pmchart Tool" on page 54 for an extensive description of the capabilities of *pmchart*.

Customization is centered on PCP views that may be created interactively and saved via the View Save option in the Views 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) reference page.

## Archive Logging Customization

The PCP archive logger is presented in Chapter 7, "Archive Logging," and documented in the pmlogger(1) reference 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* start-up options.

*/var/pcp/config/pmlogger/crontab*
: Prototype *crontab* entries that may be merged with the *crontab* for root to schedule the periodic execution of the archive log management scripts, for example, *cron.pmdaily*.

*/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, *cron.pmcheck*. Archive folios may be processed with the *pmafm* tool.

## Inference Engine Customization

The PCP inference engine is presented in Chapter 6, "Performance Metrics Inference Engine," and documented in the pmie(1) reference page.

The following global files and directories influence the behavior of *pmie*:

*/var/pcp/config/pmie/control.master*
: Control file that defines how many concurrent invocations of the various *pmie* action methods are allowed.

*/var/pcp/demos/pmie/**
: Example *pmie* rules that may be used as a basis for developing local rules.

*/usr/pcp/bin/pmie/**
: Scripts that implement the default *pmie* action methods.

*/var/pcp/config/pmrules/**
: The template masters used by *pmrules*.

## Snapshot Customization

The PCP snapshot production facility is presented in "Making Snapshot Images From Archive Logs" on page 145 and documented in the cron.pmsnap(1) reference page.

The following global files and directories influence the behavior of *cron.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 *cron.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* for root schedule the periodic execution of the archive log management scripts, for example, *cron.pmsnap*.

*/var/pcp/config/pmsnap/Summary.html*
: An example HTML page suitable for publishing images from the *pmsnap* examples via a Web server.

## 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 "The pmgadgets Command" on page 69 and the pmgadgets(1) reference page.

## 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, "System Performance Visualization Tools," and the pmview(1) reference page.

## 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 pmLoadNameSpace(3).

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 pmnscomp(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.

### 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 9-1. The correct pathname for the rightmost non-leaf node is `cpu.util`, not `root.cpu.util`.

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.

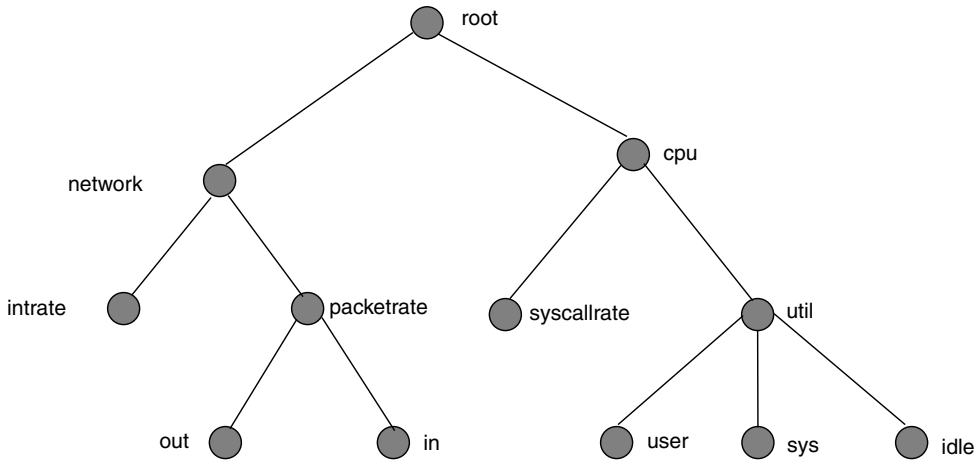


**Figure 9-1** Small Performance Metrics Name Space (PMNS)

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.

For complete documentation of the PMNS and associated utilities, see the pmns(4), pmnscomp(1), pmnsadd(1), pmnsdel(1), and pmnsmerge(1) reference pages.

## Example PMNS Specification

The PMNS specification for Figure 9-1 is as follows:

```
/*
 * PMNS Specification for Figure 4-1
 */
#include <stdpmid>
root {
    network
    cpu
}
#define NETWORK 26
network {
    intrate    IRIX: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
}
```

## PMDA Development

The 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, and when installed all of the relevant files reside in directories (one per PMDA) below the */var/pcp/pmdas* directory.

## PCP Tool Development

The 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 file reside in */var/pcp/demos/pmclient*.

*Appendix A*

# Acronyms

This chapter provides a list of the acronyms used in the Performance Co-Pilot documentation, help cards, reference pages, and user interface.

**Table A-1** Performance Co-Pilot Acronyms and Their Meanings

| Acronym | Meaning |
|---|---|
| DSO | Dynamic Shared Object |
| IP | Internet Protocol |
| PCP | Performance Co-Pilot |
| PDU | Protocol Data Unit |
| PMAPI | Performance Metrics Application Programming Interface |
| PMCD | Performance Metrics Coordination 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

## G



## H



## I



## M



## N



## O



## P

**U**



**V**



**X**



**Y**

## Tell Us About This Manual

As a user of Silicon Graphics products, you can help us to better understand your needs and to improve the quality of our documentation.

Any information that you provide will be useful. Here is a list of suggested topics:

- General impression of the document
- Omission of material that you expected to find
- Technical errors
- Relevance of the material to the job you had to do
- Quality of the printing and binding

Please send the title and part number of the document with your comments. The part number for this document is 007-2614-003.

Thank you!

## Three Ways to Reach Us

- To send your comments by **electronic mail**, use either of these addresses:
  - On the Internet: techpubs@sgi.com
  - For UUCP mail (through any backbone site): *[your_site]*!sgi!techpubs
- To **fax** your comments (or annotated copies of manual pages), use this fax number: 650-932-0801
- To send your comments by **traditional mail**, use this address:

  Technical Publications
  Silicon Graphics, Inc.
  2011 North Shoreline Boulevard, M/S 535
  Mountain View, California  94043-1389