CoMon: A Mostly-Scalable Monitoring System for PlanetLab

KyoungSoo Park and Vivek S. Pai
Princeton University

Abstract

CoMon is an evolving, mostly-scalable monitoring system for PlanetLab that has the goal of presenting environment-tailored information for both the administrators and users of the PlanetLab global testbed. In addition to passively reporting metrics provided by the operating system, CoMon also actively gathers a number of metrics useful for developers of networked systems. Using CoMon, PlanetLab administrators and users can easily spot problematic machines, where the problem may arise from the machine itself, local configuration/environment problems, or the workload running on the machine. Furthermore, users can easily observe many properties of all of the experiments running across multiple PlanetLab nodes, facilitating not only their own experiment monitoring and debugging, but also helping scale the task of finding PlanetLab problems.

In this paper we describe CoMon's design and operation, including what kinds of data are gathered, the scale of the processing involved, and the approaches we have taken to keep CoMon running. Our goal is not only to illustrate the kinds of problems faced in this environment, but also to invite others to participate, either by experimenting with the data generated by CoMon, or by building on the CoMon system itself.

1 Introduction

CoMon is an extensible monitoring system designed specifically to monitor activity on the PlanetLab global testbed. At any given time, CoMon collects and reports statistics on roughly 400-450 active PlanetLab nodes, and 200-250 active experiments running on PlanetLab. Reporting is done via a Web interface that provides the ability to sort data, run moderately complex selection queries, and show graphs of recent history. In terms of the data collected, CoMon fills a monitoring niche between the general-purpose, passive reporting tools provided by the operating system and application-specific reporting systems.

CoMon has served a number of different roles in the PlanetLab community since its launch in August 2004, including the following:

- **"Sufficient" monitoring** – For many PlanetLab users, CoMon provides enough monitoring to eliminate or reduce the need for experiment-specific monitoring. Especially for projects where development resources are tight, and the most important concerns are processes running out of control, CoMon's reporting is often sufficient to track how the experiment is consuming resources globally.

- **Community-aided problem identification** – For more attentive PlanetLab researchers, CoMon often provides enough information to determine why some PlanetLab nodes may be acting strangely, or which other experiments may be impacting their experiments. This kind of information has been extremely valuable for preserving the privacy of experiments while still enabling the overall community to help the PlanetLab operations staff in identifying problems.

- **Login troubleshooting** – The CoTest tool uses CoMon-provided data to help users determine why their logins may be failing. It tests a variety of common conditions that often cause login failures, allowing users to provide more information when reporting failures to PlanetLab support.

- **Node selection** – A relatively recent feature of CoMon allows users to specify arbitrary queries to select a set of nodes. These queries can also output results in various text formats, enabling the use of CoMon in scripting systems. This approach is useful both for identifying problems as well as for selecting nodes when deploying new (short-lived) experiments.

Over time, CoMon has grown across many dimensions, from the number of tests it runs, to the variety of output data it can generate, to the types of problems it can help find. Its mission has also grown in that time, from a purely passive Web-oriented system that only monitored node-oriented data, to one that now maintains a history of resource usage information of every experiment on every PlanetLab node.

In the rest of this paper, we will discuss how CoMon works, what it measures, and how its design is tailored to efficiently managing large volumes of data on relatively simple hardware. We also discuss current limitations of CoMon, and how it could be expanded to become more useful. In Section 2, we provide some background on CoMon, and present its design in Section 3. We discuss CoMon's operation in Section 4 and compare with related work in Section 5 before concluding in Section 6.

Figure 1: Screenshot of the CoDeeN monitoring system, which was the model for CoMon. Rows are PlanetLab nodes, and most columns contain two data values, with the column headings indicating the metrics being displayed. The bottom of the window shows a 2-day history of any cell value. Other windows can show histories on all nodes for a given metric, or histories of all metrics for a given node.

2 Background

CoMon's inspiration was the service-specific monitoring system for the CoDeeN Content Distribution Network [**?**]. At the time of CoMon's development, CoDeeN ran on approximately 110 North American PlanetLab nodes, was operational for over a year, and was receiving millions of requests daily. Part of the challenge of keeping it continuously operating was determining what sort of events could impact its performance. So, in addition to reporting the peering status, traffic rates, and user counts for all CoDeeN nodes, the monitoring system also provided status information about the PlanetLab node themselves.

A screen shot of CoDeeN's monitoring system is shown in Figure 1, and shows a number of metrics as columns, with PlanetLab nodes as rows. The sort order can be changed by clicking on a column name, and clicking on any cell value displays a historical graph for that value in the bottom window. Cell values are color-coded to indicate unusually high or low values. While CoDeeN's monitoring service does not try to be a general-purpose monitor, and presents less node-level information than the Ganglia system [**?**] (also deployed on PlanetLab), it attracted a number of users who used it primarily to gauge PlanetLab node health instead.

We attribute this usage to a number of design factors that we retain for CoMon. The most important factor, in our opinion, is the use of relatively plain HTML tables as the default mechanism for displaying data. This approach is less visually impressive than Ganglia's use of graphs to display the recent history of each metric, but we perceive several advantages in this approach. Tables efficiently utilize valuable screen real estate, making it easy not only to see the values for many nodes on screen at the same time, but also to see multiple metrics for each node. On a typical screen, a CoDeeN-style format can display 10 rows of data, with 17 columns and 2 values per column, for a total of 34 data items per node. The variety of items is important, because the diversity of experiments running on PlanetLab can have a wide variety of effects on the nodes, so seeing multiple metrics simultaneously can ease problem identification. The lower transmission size of tables also reduces the bandwidth required to see the entire summary and makes the system feel more responsive.

Several factors influenced our decision to make CoMon a separate entity from the CoDeeN monitoring system. The first was that many nodes, particularly in low-bandwidth areas, were not targets for a CoDeeN deployment at that time (though since that time, CoDeeN has expanded to run across all of PlanetLab). Having CoMon would enable us to monitor these nodes. A second factor was an ongoing discussion about the importance of privacy on PlanetLab and whether researchers should be able to see what other experiments were doing via tools such as `top` or `ps`. At this point in PlanetLab's development, resource isolation was not fully developed, so well-behaved experiments often suffered from poorly-implemented ones. Complicating matters, PlanetLab uses vservers [6], which provides the appearance of running

```
CTX  TX1 TX15  RX1 RX15 #PR PMEMMB VMEMMB %CPU %MEM NAME
516  430  933 4715 1066   5   22.0   34.2 39.3  2.2 nyu_d
610    0    0    0    0  14   22.8   41.1 25.0  2.2 arizona_stork
613  391  558  388  559  58   95.3  160.2 12.5  9.4 princeton_codeen
713    0    0    0    0  10   50.7   83.5  3.6  5.0 irb_snort
594   10   12   10   12  60   49.9  142.6  1.8  4.9 princeton_coblitz
503    1    0    0    0  23   80.0  381.1  1.8  7.9 pl_netflow
518    0    0    0    0   5    3.1    7.4  1.8  0.3 nyu_oasis
598    0    0    0    0   3    6.5   11.9  0.0  0.6 uw_ah
738    0    0    0    0   3    6.2   12.0  0.0  0.6 purdue_4
596    0    0    0    0   5    3.5    9.2  0.0  0.3 ucb_pier_2
726   10   10   51   38 133   68.7  112.1  0.0  6.6 mit_app
577    0    0    0    0  11    9.5   21.6  0.0  0.9 upenn_maoy
636    0    0    0    0   6    8.6   27.1  0.0  0.8 mit_dht
640    4    4    5    4   6   16.8  232.2  0.0  1.7 ucb_srhea
760    0    0    0    0   4   42.0  608.9  0.0  4.1 harvard_sbon_test
  0    0    0    0    0  39   25.1  228.1  0.0  2.4 root
[...]
```

Figure 2: Excerpt from sample output from the slice-centric daemon, showing resource consumption per slice. The columns, from left to right, are context number (numeric userid for the slice), transmit and receive rates for the past 1 and 15 minutes, number of processes, physical and virtual memory consumption, overall CPU and memory utilization, and slice name. Most nodes have on the order of 50 slices running at a time.

on a virtual machine, without providing full resource isolation [1]. As a result, while users could see that a node was behaving strangely from its responsiveness or other factors, they had no recourse but to notify PlanetLab support, who could then try to investigate. As PlanetLab was growing, this approach of relying on PlanetLab Support to police all experiments running on all nodes was not scalable.

For the sake of clarity, we define some terms used in the rest of this paper. We use the term *node* to refer to any PlanetLab machine, and *site* to refer to all nodes at a given physical location. Each PlanetLab account, which can be instantiated on any and all of the PlanetLab nodes, is called a *slice*, and its instantiation on one node is called a *sliver*.

CoMon was viewed as a privacy compromise, where anyone could see resource consumption details about other slices, without having access to the information on a per-process basis. This approach would also allow the entire PlanetLab community to get involved in policing resource usage, and to help spot problems on nodes. By making resource usage public information, it was hoped that communal pressure could be exerted on those slices that were behaving poorly. PlanetLab's support staff could obviously get involved at any point, but the goal was to reduce the need for them to have to identify the problem from scratch every time.

Additionally, by expanding the amount of information available about node health, CoMon and other services could be developed in their own slices. This approach is preferable to running everything inside the root context, because not only does it provide the kind of isolation that running unprivileged processes provides, but it also motivates a more decentralized design. As more infrastructure gets built by the community, the fewer feature requests need to be handled by PlanetLab's central developers.

[1] The use of vservers provides a level of scalability and performance not easily achieved with other production-level virtual machine techniques, so for Planetlab, its benefits outweigh its drawbacks

3 Design

In this section, we describe the overall design of CoMon, as well as the design of the individual pieces. At a high level, CoMon consists of two daemons that run on each PlanetLab node, a centralized data gathering and processing infrastructure, and a display facility that provides support for simple user-specified queries. We describe each of these in the rest of this section.

3.1 Per-Node Daemons

CoMon relies on two daemons running on each node, which provide node-centric details as well as slice-centric details. Both daemons accept HTTP requests and respond with HTTP responses, to allow them to be accessed from Web browsers in addition to being used with automated systems. Both daemons provide responses in human-readable text rather than in binary, mostly to allow easy extensibility.

3.1.1 Slice-centric daemon

Of these two daemons, the information presented by the slice-centric daemon is much simpler – it reports the aggregate resources used by all processes within each slice. It reports 11 metrics – the transmit and receive bandwidths for the past 1, 5, and 15 minutes, the physical/virtual memory consumption, the CPU and memory usage, and the number of ports in use. This daemon is an extension of the slicestat sensor server originally developed by Brent Chun.

This daemon gets most of its data from the slicestat sensor operating on each node, and formats it in a manner similar to the "top" monitoring tool. The daemon has a main event-driven process that responds to client requests, and a second helper process that communicates with the slicestat sensor in order to get the data it needs. All outstanding requests can be

satisfied by a single response from the slicestat sensor, since the same data is presented to all clients. A sample of this sensor output is shown in Figure 2.

3.1.2 Node-centric daemon

The node-centric reporting currently covers 57 values, which consist of OS-provided metrics, values that are passively measured or synthesized from other sources on the node, and values that are actively measured by means of test programs running on the node. A brief summary of these metrics is provided below:

- **OS-provided** – uptime, CPU utilization (overall and system), memory size, active memory consumption, disk size, disk space available, swap size, swap space available, date, 1 minute load, 5 minute load, swap in/out rate, and disk read/write rate.
- **Passively-measured/synthesized** – last time ssh succeeded, last time slice-centric daemon succeeded, clock drift, number of raw ports and icmp ports in use, number of slices in memory, number of slices using CPU, number of ports in use, number of ports in use for more than an hour, and resource hogs (which experiments are using the most CPU, memory, processes, bandwidth, and ports)
- **Actively measured** – max/avg value reported by a 1-second timer, max/avg value for time needed to make loopback connection, whether the global file table has free entries, amount of CPU available to a spin-loop, amount of memory pressure seen by a test program, UDP and TCP failure rates for local DNS servers, last HTTP failure time, detecting presence of transparent Web proxies

The guiding principle for choosing metrics for inclusion is whether they can provide insight into why a researcher's experiments may be behaving strangely on a given node. This approach allows us to prune the dozens of OS-provided metrics to a more manageable set, and it also guides the development of some of the more unusual active measurements. For example, the CPU consumption of a spin loop [2] and the observed behavior of a timer both give some insight into how much CPU PlanetLab's custom scheduler gives each experiment. Likewise, the transparent proxy test, which checks if a remote server sees the same IP address as a local client sees, is useful for experiments that contact HTTP servers, but may not be useful in non-testbed environments.

The structure of the node-centric daemon is intentionally simple – it consists of one main event-driven process that spawns a pool of helpers, as a Web server might handle different CGI programs. Each helper process is persistent, and periodically generates updated values for the quantities it measures. When a client requests data from the node-centric daemon, it receives a response that combines the most recent values from each helper. This approach leads to a large number of processes (over 100 with the current design), but most processes are waiting at any given time and have small footprints. On average, CoMon uses less than 0.5% of each node's CPU for its processing. A sample node-centric response is shown in Figure 3.

[2] This test was original conceived by Andy Bavier to gauge PlanetLab capacity in the run-up to the SigComm 2005 deadline.

```
VMStat: 2 1 148384 20916 44904 440904 1 0 [...]
CPUUse: 68 100
DNSFail: 0.0 0.0 0.0 0.0
DfDot: 14% 155.504 179.39
Date: 1133129457.852773000
Uptime: 96899.85
Loads: 5.09 3.75 3.88
Timer: 236.646000 11.412840
FdTestHist: 0x0
ServTest: 12.505000 1.617917
MemInfo: 0.987202 60.7348 14.1503
KernVer: 2.6.12
Burp: 28.5%
MemPress: 98
LastSsh: 1133128201
Test206: 206   0   0   0
SamePorts: 457 list_sameports
SnapPorts: 1469 list_portsnap
SameHog:     178 princeton_codeen
SnapHog:     632 princeton_codeen
RawPorts: 6
ICMPPorts: 35
CPUHog: 71.7 nyu_d
MemHog: 14.8 princeton_comon
TxHog: 939 nyu_d
RxHog: 567 princeton_codeen
ProcHog: 60 princeton_coblitz
NumSlices: 43 0.000000
LiveSlices: 9
```

Figure 3: Sample output from the node-centric daemon showing all measurements performed. The interpretation and display of the measured values is controlled via a configuration file on the machine that gathers the data from all of the per-node daemons.

3.2 Data Gathering

While the CoMon daemons operate on each node, the bulk of CoMon data gathering and processing operates in a centralized fashion, mostly to reduce complexity and to ensure that we have a properly-provisioned machine available.

Data is collected from the per-node daemons every five minutes, and is stored in a variety of files. Most nodes (typically over 90%) respond within one second, and generally fewer than five nodes take more than 10 seconds to respond. All fetches are performed in parallel to reduce latency. The raw data, in text format, is stored in several places: (1) a file containing the most recent snapshot for all nodes, (2) appended to the end of a file containing all of the data gathered for the current day, and (3) also written into one file per live node so that the last live data is available for post-mortem node analysis.

The most basic processing of the raw data generates

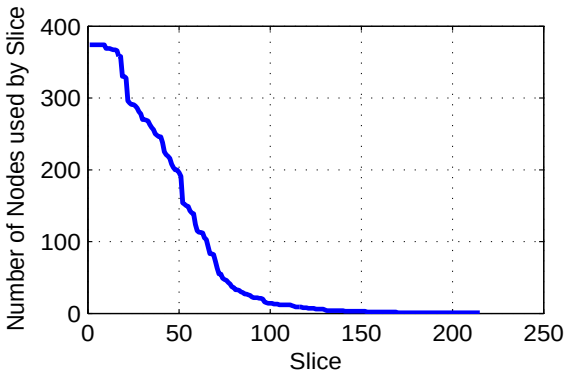

Figure 4: Snapshot of # of nodes used by each slice on 11/28/2005

metafiles that can then be used by CGI programs to generate sorted HTML tables on demand. The number of these metafiles is O(slices + nodes) – the slice-centric information is shown for all the slices on each node, and for all of the nodes for a particular slice. Additionally, summaries are generated with the slice-centric information, showing the maximum, average, and total values for each of the slice-centric data items across all nodes in a slice. Only two metafiles are generated with the node-centric data – one that contains all of the node-centric data for each node, and another that omits some of the less-important fields, such as identifying all of the resource hogs. The second metafile is intended to be used on narrower screens, to avoid horizontal scrolling.

From a scalability perspective, the processing steps discussed above are quite tolerable, since the number of individual files generated is roughly # slices + 2 ∗ # nodes. At PlanetLab's current size and usage (approximately 600 nodes and 250 experiments), these steps generate 1500 files every 5 minutes, or 5 files per second. If we assume these files are randomly-accessed, and if we have 2 other seek-causing metadata accesses per file, this rate still yields only 15 seeks/second, which is at least a factor of 5 lower than current disks can handle.

In terms of network and disk bandwidth, the current numbers also leave plenty of headroom. CoMon currently generates 600 MB of raw data per day taking into account both the node-centric and slice-centric fetches. On average, this is less than 7.5 KB/sec. Compression can reduce this number further – CoMon's daily logs, when compressed with `bzip2`, generally drop to 10% of their raw size.

3.3 Scaling Processing

The scalability issues for CoMon arise from the desire to present graphical two-day histories for any value shown in the tables. We use the popular RRDTool [5], which is designed to efficiently maintain time-series databases for monitoring applications, from which monitoring graphs can be easily generated. We have one separate database file for each row of any table, which provides us the best practical balance between efficiency and flexibility. This approach allows us to add nodes/slices to CoMon without having to reformat the database files. Adding more metrics per row, however, does require updating the database files. While RRDTool has recently added support for transferring values from one database to another, we currently just erase old database files when the format changes.

The total number of database files is then O(# nodes + 3 ∗ # slices + # slivers), where a sliver is a slice instantiated on a node. In theory, the maximum number of slivers is # slices ∗ # nodes, but not all slices are deployed on every node. In practice, about 25 slices (out of 200-250 slices) are deployed across more than three-fourths of all nodes, while another 25 are deployed on more than half. This kind of distribution can be seen in Figure 4, which shows a fairly typical snapshot of the number of nodes used by each slice. In total, of the 100,000 possible slivers that could typically exist, about 20,000-22,000 exist at any given time in the recent past. A longer history of the number of total slivers is shown in Figure 5. The dip at 120 days corresponds to a major PlanetLab upgrade that required significant downtime. The drops near 170 days were due to problems encountered in the process of testing a kernel upgrade. Other smaller bumps are typically the activity near conference submission dates.

If these databases are to be updated every 5 minutes, the number of seeks involved exceeds the rate that can be achieved by current disks. Assuming an optimistic number of 3 seeks per file, this workload would require 3 high-performance disks running continuously at maximum speed just to maintain the current update rate. In practice, we seem to require more than 3 seeks per file (understandable, given the number of directories involved), and other activities also use the disk. Even if this seek rate was achievable, it would be dangerous from a design standpoint to be so close to the hardware limits at all times. Not only would it limit scalability (any new slices might cause overload), but even a slight miscalculation could cause each update to run slightly above the time allotted, cumulatively increasing the overall load on the system and eventually causing collapse.

The approach we take to this problem stems from the observation that higher disk bandwidth is easier to achieve than higher seek rates, and that preserving disk locality via batching can exploit the higher bandwidth. Batching also naturally implies delaying some operations, possibly causing longer delays, so a naive approach can result in lower perceived performance. To reduce the perceived impact of batching while still using it to address the disk seek problem, we split the data processing into several steps, and batch independently at each step. In this manner, we can control how much delay is introduced in the various tables/graphs, and we can control the resource usage of the various components. The four processing steps are described below.

- **Gather Data** – Every 5 minutes, gather data from the slice-centric daemons. Write all data sequentially into a snapshot file of the current state, write another copy into an "update" file, and also append it to the history file for the current day. If no other instance of the pro-

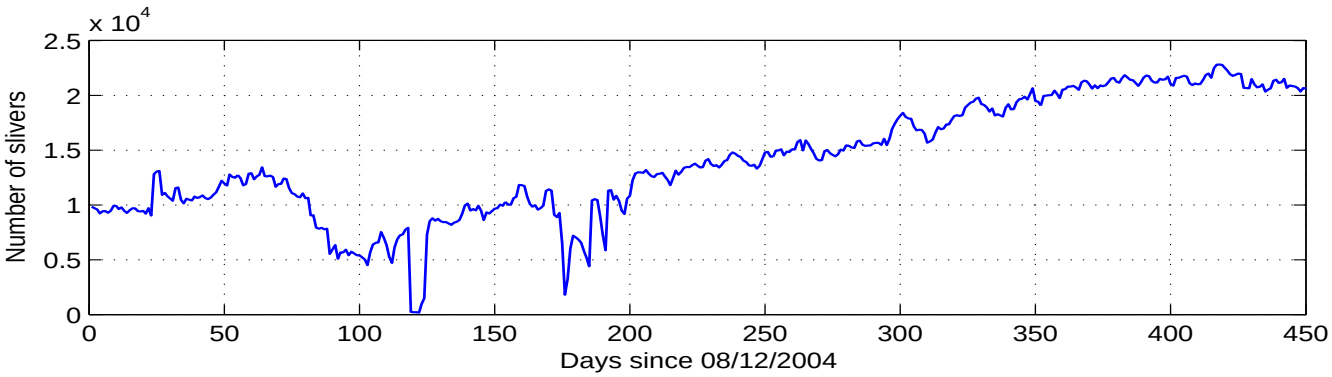

Figure 5: Daily count of total number of slivers

gram is currently writing tables, write the per-node tables, the per-slice tables, and the tables for the maximum/average/total resources consumed by each slice.

- **Split** – This program checks for the "update" files generated by the "gather data" process. It takes the monolithic update file and splits it into a separate file per slice. It repeats the process for every monolithic update file it finds.
- **Update Stats** – Using the per-slice update files, this process updates the RRD databases containing the per-slice statistics for maximum, average, and total resource consumption. Only one instance of this process runs at a time, and processing is comparatively CPU intensive. On a fast dual-processor node, this step takes only 10-20 seconds, but can take minutes on a more loaded uniprocessor. With these times and the dependent processing steps, the *graphs* for the per-slice statistics tend to lag no more than 10-15 minutes behind the current time. The tables, in contrast, are updated in real time with the latest gathered data.
- **Update Slivers** – Takes the per-slice update file and splits it by node to generate per-sliver files. It then takes all of the per-sliver files and updates the corresponding RRD databases. On a fast dual-processor machine with two hard drives in a RAID0 configuration, this step takes 150-250 seconds. On a slower uniprocessor, this step can take over 2000 seconds.

The benefit of this partitioned approach is that we can optimize (to the extent possible) the amount of lag seen by users [3]. The most commonly-used portions, the node-centric and slice-centric tables are always up to date with respect to the most recent data gathered. The graphs for the node metrics are also recent, and only on slow systems will the slice summary graphs be delayed. Finally, the graphs for the sliver metrics, which consume the most resources and are the least-frequently used portion of the system, can experience the most lag.

[3]To clarify, we use the term lag to define the time difference between the most recent value in the system versus the current wall-clock time

An interesting additional benefit of this approach has been the ability to tune the lag to reduce heat-induced stress on the disks. We have found that 1U rack-mounted systems often cannot efficiently dissipate the heat from drives running at full seek capacity, and that the overheating drives cause regular system lock-ups ranging from several seconds to over a minute at a time. By adding delay loops into the processes that update the RRD databases for the slice summaries and slivers, we can reduce the heat buildup in these systems for the cost of some extra lag in the graphs.

3.4 Node Selection

As CoMon's utility increased, so did many users' desires to do something with CoMon's data, which led to the development of node selection support. Originally, CoMon users had several less-than-appealing options when trying to incorporate CoMon into other systems: they could screen-scrape the HTML tables, they could directly query the sensors, or they could copy-and-paste data from their browsers. Some mechanisms was needed to automate this process and leverage the support CoMon already had developed. This part of the design had several guiding principles: keep the process as simple as possible, make it easily accessible to people familiar with C, allow it to be easily scriptable, and reduce the "buy-in" necessary for users to adopt the solution.

We ultimately chose to extend the table metafile that CoMon generates to be used with the CGI application that generates all CoMon tables. By doing so, we add virtually no overhead to the process of generating regular CoMon pages, making the infrastructure change relatively transparent. We pass the selection criteria in the URL itself, making it easy to be used both with browsers as well as standalone downloading tools such as `curl` and `wget`, popular in scripts. We opted for a simple, C-like syntax (with matching operator precedence) for specifying nodes, since it was compact and would be easily recognizable to most of our target audience. While these choices are not ultimately as expressive as other options, we felt that they presented a good balance between simplicity and utility.

When the node selection support is used, each row is tested

against the selection criteria, and if the statement evaluates positively, the node is included in the display. Column names are treated like variables, and the standard set of comparison operations are provided. To get a sense of how this selection works, the following text can be added to the node-centric table's URL to select lightly-loaded nodes only:

select='resptime $>$ 0 && 1minload $<$ 5 && liveslices $<=$ 5'

Specifying a nonzero response time selects only live nodes, while specifying a 1 minute load of less than 5 and fewer than 5 slices actively running further restricts it to only those nodes that have relatively little work competing for the CPU [4]. The selection criteria can also be used to identify problematic nodes, such as this example:

select='drift $>$ 1m || (dns1udp $>$ 80 && dns2udp $>$ 80) || (resptime $>$ 0 && gbfree $<$ 5) || sshstatus $>$ 2h'

This selects for four types of problems – clock drifts greater than 1 minute, primary and secondary DNS failure rates above 80%, live nodes that are running short on free disk space, and nodes where the SSH daemon has been refusing connections for over 2 hours. This kind of selection statement would be useful to groups like PlanetLab's operations staff in order to identify problematic nodes, and perhaps show them on a Web page. However, when we combine the two selection categories together, we can find nodes with low load and without problems:

select='(resptime $>$ 0 && 1minload $<$ 5 && liveslices $<=$ 5) && ((drift $>$ 1m || (dns1udp $>$ 80 && dns2udp $>$ 80) || gbfree $<$ 5 || sshstatus $>$ 2h) $==$ 0)'

Users can further specify the directive `format=nameonly` in the URL to specify that instead of generating HTML tables as output, CoMon should produce only a list of node names in text format. This kind of URL is then easily amenable to use in scripts.

The additional code to support these features consists of a simple parser and evaluator, and some additional support when generating the table metafiles. In total, this node selection code amounts to less than 300 additional semicolon-lines added to the system.

The computational cost of this support is relatively modest, and is actually a net benefit for the interactive user, since the extra latency in performing the selection is almost always recovered by the lower time required by the browser to render fewer rows. Some sample times measured at the server are provided in Table 1 for the types of selection statements discussed above. Note that while using the node selection support increases runtime, more complicated statements that reduce the amount of data produced can take less time than simpler statements.

[4] Since PlanetLab is a shared infrastructure, higher load levels are not uncommon

Test Name	Time	Output Size
base	90ms	2772 KB
+ live	101ms	2331 KB
+ low load	97ms	405 KB
+ no problems	99ms	394 KB
+ names only	95ms	2 KB

Table 1: Times and output sizes resulting from various selection options.

4 Discussion

While an open-ended discussion about the role of monitoring in distributed systems is beyond the scope of this paper, a more focused discussion about CoMon can be shaped by two questions: has it proven successful, and will it keep working. We address these questions below.

4.1 Does CoMon Work?

Though we obviously cannot quantify the impact of CoMon in a controlled manner, we can present some evidence of the kinds of benefits CoMon is producing for PlanetLab. While all of the examples are anecdotal, we believe that the number of them and their diversity serve to illustrate CoMon's impact.

Two tools have used CoMon's output to diagnose problems logging into PlanetLab – one developed by Neil Spring called `why` and our own tool, called `CoTest`, which was inspired by Neil's script. Both tools are available as standalone programs that run locally on a user's machine. They download data from CoMon and other sources, and determine if several common problems are preventing users from accessing the node. These tools serve as a self-help mechanism before asking PlanetLab support to investigate login problems.

The developers of OpenDHT [12], a notable and widely-used service running on PlanetLab, used CoMon to identify where problems were arising in their service. OpenDHT provides a publically-accessible distributed hash table across PlanetLab, and relies on an event-driven server process. In normal operations, this server should consume CPU only when data is exchanged with other computers, but in exceptional conditions, it was needlessly spinning. Its extra CPU consumption did not affect the correctness of its operation, so it was not initially noticed by its developers. With CoMon showing that it was one of the largest consumers of CPU cycles on PlanetLab, its developers were able to find the nodes where problems were arising and track their causes. Fixing the problem eliminated the spinning, improved OpenDHT's latency, and brought its per-node CPU consumption down to 1-2%.

In another case, CoMon was used to determine which experiment was freezing dozens of PlanetLab nodes. In September 2004, as the NSDI paper submission deadline approached, blocks of PlanetLab nodes became completely unresponsive in a few days. Dozens of nodes were dying at nearly the same time, and the machines were wedged to the

Figure 6: Screenshot of CoMon's node-centric view, sorted by the Live Slices column. The icons near the node names allow for post-mortem slice snapshots and current resource usage.

point that only reboots were able to restart them. However, with no ability to log into the nodes, all the machines required either remote power cycles via power control units, or manual resets performed by on-site administrators. Using CoMon's snapshots of slice activity for each dead node, we were able to identify the slice that was responsible and the mechanism that was causing the nodes to freeze. The slice was quickly allocating large amounts of memory, exhausting swap space and triggering a kernel lock-up on the nodes. The problems disappeared when the slice stopped running on other nodes. Without CoMon, the problematic slice would not have been easily identified, and the situation might have worsened closer to the NSDI deadline.

In response to the swap memory exhaustion problem, Andy Bavier developed a monitoring program, pl_mom, that uses CoMon to help police swap usage. When a node's swap memory usage crosses a given threshold, pl_mom uses CoMon to determine which sliver is using the most physical memory, and kills it. While it would be desirable to determine which sliver is using the most swap, this accounting is unavailable in Linux, and physical memory consumption is often highly correlated to swap consumption in PlanetLab. As a precaution, if swap usage continues to grow after killing the largest memory consumer, pl_mom reboots the node. While this option is undesirable, an automatic reboot is preferable to the node freezing and requiring manual intervention.

A final piece of evidence illustrates our belief about CoMon being sufficient for a number of monitoring tasks. We have repeatedly received requests for CoMon to include per-slice disk usage in its slice-centric information. Disk space is generally quite available on most PlanetLab nodes, and is not much of a contended resource, since every slice has its own 5GB quota. Having individual experiments monitoring their own disk space using any sort of parallel ssh tool would be quite trivial. However, by having this value available in CoMon, users could more easily monitor their own experiments and retain the benefits of history and logging that CoMon provides. We have not yet been able to include this information because CoMon, running in its own vserver context, does not have the privilege to view the disk consumption of other slices. However, we are in the process of receiving the necessary information via the Proper service [10].

4.2 Does CoMon Scale?

It is reasonable to ask whether the approach we have taken is the right approach to make a large-scale monitoring system scale. Given our experiences and the measurements of the current system, we are fairly confident that our current design can handle a factor of ten expansion in PlanetLab without much concern. The parts of the monitoring system that run on a fixed schedule, like the data gathering, are not a scalability issue. The sliver processing automatically adjusts its frequency based on the amount of work needed, so PlanetLab's growth will only cause an increased lag in the per-sliver graphs. If the number of slivers increases by a factor of 15, the lag in the sliver graphs would increase to one hour, assuming our current hardware stays fixed (dual processor 3.2

GHz Xeon, two SATA drives in a RAID0 configuration).

Another option for coping with scale is to add more laziness into the data processing, ideally avoiding most work until a graph or table is requested. Even at our current scale, we can consider whether per-sliver database updates can be done lazily. The current graphs in CoMon display data for 48 hours, so if no data from a sliver is graphed in 48 hours, then all database updates performed in that time are simply wasted work. Otherwise, these updates can be viewed as eager updates which reduce the latency involved in producing the graph on demand.

The difficulty in this approach would be determining how much work needs to be done to make the graphing latency tolerable. If only the raw data is kept and the sliver graph is built completely on demand from it, then the latency is the time required to parse two days worth of data and load it into the graphing tool. At current scale, this is slightly more than 1 GB of text, which would take 30 seconds to read at the maximum sustained bandwidth of most drives. Reading a bzip-compressed version would still take 3 seconds, but would also require a significant amount of CPU for decompression. In our experiments with compressing CoMon data, bzip2 compresses the data to almost half the size achieved by gzip, but requires more than 25 times as much CPU for decompression.

Keeping all of the raw data in main memory for the most recent two days is technically possible, and would eliminate disk bandwidth issues. Without any compression, this approach would begin to break down at three times our current scale, since it would exhaust the physical memory of the machine. Beyond this point, the steep penalties of using demand-paged virtual memory would become a problem, and would likely cause this approach to be infeasible. However, some hybrid approach, especially using compression, may work for the most popular set of sliver graphs.

In the long run, we believe that some form of processing or indexing is necessary to make the per-sliver graphs usable in real time, so a purely lazy approach may never work. Finding the right tradeoffs at larger scale will continue to be an interesting area for CoMon to explore. The possible design space is large, and different approaches will have tradeoffs related to resource consumption, latency, lag, and the impact of popularity distributions. We intend to explore some of these issues in future work.

5 Related Work

The project most closely related to CoMon is Ganglia [9], which has been widely adopted for monitoring compute clusters. These systems tend to have different workloads and properties than PlanetLab – in particular, these environments tend not to have the wide swings in resource consumption seen on PlanetLab, nor do they generally have to cope with resource sharing at the scale of Planetlab experiments, so the per-sliver monitoring which causes the scalability concerns in CoMon is absent in Ganglia. While Ganglia has added support for more distributed environments, and has been running on PlanetLab, it is understandable that a general-purpose system would be less interested in this level of special-purpose support for a unique system. Ganglia does allow site-specific metrics to be added to the data it collects, but in our case, metrics such as sliver resource usage are not easily integrated into the node-centric data. This kind of data gathering is part of our motivation to make CoMon a separate entity from CoDeeN's monitoring system, rather than simply extending it.

Another monitoring system on PlanetLab, PsEPR [7] (formerly known as Trumpet), focuses on finding problems via several tests to gauge node health. The first main difference between PsEPR and CoMon is in the types of tests performed – PsEPR's tests largely test the PlanetLab infrastructure itself, while most of CoMon's tests are measuring functions of the workloads and slices. The second difference is in the mechanism used to communicate information – PsEPR uses the Jabber protocol [2], which provides a decentralized publish/subscribe communications channel. While this approach can be more scalable than CoMon's centralized approach, we have not found communications overheads to be a concern at our current scale. Even with our current approach, scaling to 10 times our current size would not consume enough centralized monitoring bandwidth to be an issue.

Two deployment/management systems, SWORD [11] and PLuSH [4], have features designed for selecting nodes and managing experiments. These can be viewed as more sophisticated forms of solutions like AppManager [3]. CoMon's node selection support performs some functions similar to these systems, but with fewer options and a C-like query language instead of an XML-based one. While the use of XML can provide more expressiveness, our personal experience suggests that CoMon's select support can be used to develop reasonably complex queries. In the future, if we find that specifying the query within the URL is too constricting, we may look at options to specify a separate URL that points to the query. However, the premise we want to test with CoMon's node selection support is whether something simple and scriptable is good enough for many purposes.

Finally, no discussion of monitoring systems can be complete without mentioning SNMP, the simple network management protocol [8]. This system is widely used and supported by commercial tools such as HP's OpenView [1]. It provides an open protocol format that can be used to monitor a variety of different types of equipment, using a vendor-supplied management information base (MIB) that provides the specifics of the kinds of monitoring provided by each piece of hardware. SNMP's hierarchical MIBs plus associated control software provide a rich environment, which we initially considered using for CoDeeN monitoring. Eventually, we determined that the MIME-header format was simpler to use, expand, and process, while still meeting our needs.

6 Conclusion

CoMon, a wide-area monitoring system for PlanetLab, is a narrowly-tailored system designed to help provide useful monitoring and debugging information for a unique worldwide platform. Because of the special nature of PlanetLab, CoMon faces a number of challenges not seen in other monitoring systems. Among these are extremely large scale for infrequently-used data, and the desire to efficiently devise active measurements that provide insight into the (mal)functioning of a node without revealing too much information about what is running on it. In the process of developing CoMon, we have learned a number of useful lessons ranging from simplicity in design, sparse but effective user interfaces, and C-friendly query mechanisms.

CoMon has proven useful in expanding the number of people who can help identify and narrow problems occurring in PlanetLab, allowing the maintenance of PlanetLab to scale as the system grows larger. In addition, CoMon has helped in post-mortem analyses of problematic events in PlanetLab, and has provided some impetus for developing automated systems that prevent their recurrence. Anecdotal evidence suggests that researchers using PlanetLab also benefit from having CoMon provide passive monitoring and historical information for their experiments. We expect that these benefits to continue as PlanetLab grows, attracts more researchers, and encounters an ever-expanding range of workloads. We also expect that monitoring such a system will present interesting opportunities and lessons in the future.

Acknowledgments

We would like to thank all of the CoMon users who have provided us with feedback about the system, and Brent Chun for giving us the source to slicestat. This work was supported in part by NSF Grants ANI-0335214 and CNS-0439842.

References

[1] HP OpenView products. http://www.managementsoftware.hp.com/products/.

[2] Jabber Software Foundation. http://www.jabber.org/about/overview.shtml.

[3] PlanetLab Application Manager. http://appmanager.berkeley.intel-research.net/.

[4] PLuSH. http://sysnet.ucsd.edu/projects/plush/.

[5] RRDTool. http://people.ee.ethz.ch/~oetiker/webtools/rrdtool/.

[6] Vserver. http://linux-vserver.org/.

[7] P. Brett, R. Knauerhase, M. Bowman, R. Adams, A. Nataraj, J. Sedayao, and M. Spinde. A shared global event propagation system to enable next generation distributed services. In *Proceedings of First Workshop on Real, Large Distributed Systems(WORDLS)*, December 2004.

[8] J. Case, M. Fedor, M. Schoffstall, and J. Davin. A simple network management protocol (SNMP). RFC 1157, May 1990.

[9] M. L. Massie, B. N. Chun, , and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation, and experience. *Parallel Computing*, 30(7), July 2004.

[10] S. Muir, L. Peterson, M. Fiuczynski, J. Cappos, and J. Hartman. Proper: Privileged operations in a virtualised system environment. In *Proceedings of the USENIX Annual Technical Conference*, April 2005.

[11] D. Oppenheimer, J. Albrecht, D. Patterson, and A. Vahdat. Distributed resource discovery on PlanetLab with SWORD. In *Proceedings of First Workshop on Real, Large Distributed Systems(WORDLS)*, December 2004.

[12] S. Rhea, B. Godfrey, B. Karp, J. Kubiatowicz, S. Ratnasamy, S. Shenker, I. Stoica, and H. Yu. OpenDHT: A public DHT service and its uses. In *Proceedings of ACM SIGCOMM*, August 2005.