CLUSMON: A BEOWULF CLUSTER MONITOR

by

Conrad Kennington

A project

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

Fall 2006

The project presented by *Conrad Kennington* entitled *Clusmon: A Beowulf Cluster Monitor* is hereby approved.

Amit Jain, Advisor	Date
John Griffin, Committee Member	Date
Paul Michaels, Committee Member	Date
John R. Pelton, Graduate Dean	Date

Dedicated to my wife, Sarah.

ACKNOWLEDGEMENTS

Thank you my pregnant wife, Sarah—who is due to have our first baby within the same week of my graduation. You've given me love and support, and motivation to finish my project on schedule! Also, thanks to my parents who ignited and fueled my passion for computers with our first IBM 286.

Kevin Twitchell, my favorite undergraduate computer science teacher from Brigham Young University-Idaho is credited with preparing me for graduate school. And once a graduate student, Dr. Amit Jain led me the rest of the way. He is my advisor and friend whose introduction to distributed computing unleashed a torrent of creativity. It was Amit who suggested Clusmon as a project.

Thank you to my part-time employer Mobile Dataforce for making it possible for me to pursue a graduate degree at a full-time pace, and still make a living.

This material is based upon work supported by the National Science Foundation under Grant No. 0321233. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

AUTOBIOGRAPHICAL SKETCH

At thirteen, Conrad Kennington started programming after milking cows on the family dairy. As a teenager, he tinkered with computers and took odd jobs as a consultant in the community as an unlikely computer geek. Conrad attended Brigham Young University-Idaho, deciding between two majors: Computer Science and Computer Engineering. It turns out Computer Science was the cheaper hobby. There he met his wife Sarah, an English major. He followed her home one summer to Houston, Texas where he took his first internship at EPS Software. They graduated together in May 2005, and moved to Boise, Idaho. Conrad met the Chair of the Computer Science Department at Boise State University, Dr. John Griffin. This conversation had phrases like "tuition assistance" and "graduate assistant", which convinced Conrad to pursue interests that a Bachelor Degree couldn't satisfy. While enrolled as a full-time graduate student, Conrad held three part-time jobs. He worked in the Computer Science Linux Cluster Lab as a graduate assistant, a software developer at Mobile Dataforce, and a computer science teacher at Stevens-Henager College. It was a foot in three doors: teacher, student, and worker in industry: three simultaneous views. It accelerated his understanding in the field of Computer Science. Through all of this, Conrad and Sarah bought a home in Kuna, Idaho, where he plans to develop software and live happily ever after.

ABSTRACT

"...he has the strength of thirty men in his hand grip. -Beowulf"

Weather forecasting, genome mapping, cryptography, special effects—these are only a handful of uses which benefit from the considerable computing power of Beowulf Clusters. Distributing workloads across a network of commodity computers makes it possible to crunch substantial amounts of data quickly, and new uses are always being discovered.

Beowulf Clusters are created from commodity hardware and managed using specially designed cluster software. There are many commercial and open-source cluster management options available, but unfortunately they have limited monitoring capabilities. Most collect and perform data analysis locally, causing a heavy burden at the worst possible place: the master node. Currently, there are no open-source monitoring systems available that offloads the monitoring analysis from the master node—except Clusmon.

Clusmon is a Beowulf Cluster monitoring system designed to move data analysis to another location, on a machine that is not part of the cluster. Displayed as a web-based dashboard of dials and graphs, it monitors CPU temperature, fan speed, CPU load, memory usage, uptime, and network traffic. Clusmon warns the administrator if hardware thresholds are exceeded, and archives historical data which can be

graphed over time. Cluster administrators will find Clusmon an indispensable tool for monitoring clusters.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Clusmon: A Beowulf Cluster Monitoring System

Clusmon is a Beowulf cluster monitoring system. It monitors nodes on a cluster and displays cluster data as a virtual dashboard of dials and graphs. Its purpose is to assist cluster administrators in monitoring the overall health and usage of a cluster, flag possible hardware failures, and store data as a historical reference.

1.2 Rationale and Significance

Donald Becker, the father of the Beowulf Cluster, said in an interview:

"Diagnostics and monitoring have always been part of using cluster, but as more clusters are deployed and the underlying operating system becomes more complex, there is an increasing need for tools that identify the problem and point to their cause. For the future I see the focus continuing on making the software easier to use." [1]

Clusmon is a fulfillment of that prediction. Its designed to be easy to install and use. The front end is a dynamically generated web page, which is easily readable and aesthetically pleasing. It comes at a time when cluster use is rapidly growing and

options are limited.

1.2.1 Clusmon Features

Clusmon features extracted from the original project proposal:

- Fast, efficient, robust, reliable, and maintainable
- Ability to monitor: CPU temperature, system temperature, fan speed, uptime, memory usage, network traffic, CPU load, jobs running, and batch processes
- Offset data analysis to a web server
- Web interface to allow remote monitoring
- Database for historical graphing
- Automatically send a warning email if the cluster temperature exceeds thresholds
- Automatically shutdown a node if the temperature exceeds thresholds
- Easy to install, set-up, and configure

1.2.2 Clusmon vs Existing Cluster Monitors

Clusmon is original in its own right. It's the only cluster monitor to date that uses a middle-end database with a front-end web interface. This unique design allows it to function perfectly even while the cluster is heavily loaded. Most other cluster

monitors are built into the management software and remain resident on the master node. The disadvantage here is the user must be at the master node to use the software, and monitoring capabilities can stagger as the cluster is being used.

The performance of Clusmon's web interface never wavers. It gets its data from a database, and the database always has data. Since the data is supplied by the master node, only the master node and worker nodes will notice the performance hit. But the user at the front-end doesn't notice. Additionally, a web interface allows the user to remotely monitor the cluster from any computer with Internet access.

Another useful feature of Clusmon is the ability to keep information in a database for historical purposes. Clusmon data is archived, and can be displayed on a line graph over a period of months.

There are several other existing solutions, both open-source and commercial. Some of the more popular ones are listed below:

Clusterprobe [2] written by Z. Liang, Y. Sun, C. Wang, closely resembles Clusmon. It's an open-source cluster monitor written in Java that uses RMI (remote method invocation) for communication. It's different in that Clusterprobe divides the cluster into subsections, with one node in the subsection acting as a "monitoring proxy." There is no indication that Clusterprobe collects temperature data.

ClusterWorX [3], by Linux Networx, is a commercial, all-purpose cluster management tool. The monitoring feature has the ability to collect data at an extremely fast rate. An administrator would need to be physically at the cluster to take advantage

of this sample rate in real time. While it's a very robust solution, it is also very expensive.

Ganglia [4], by the University of California, Berkeley, is probably the most popular and widely used open source cluster monitor. Clusmon is like Ganglia in that they both use a multicast for registering new nodes, making it possible to maintain a dynamic list of nodes to monitor. However, one major difference is that Ganglia uses XML for transferring much of the cluster data, while Clusmon uses binary data structures.

Supermon [5] is an open source cluster monitor put out by the Advanced Computing Laboratory. It allows high sample-rate monitoring to be periodic or reactive, meaning it only returns changed data. Supermon also doesn't pass binary data, but rather s-expressions which resemble lists in LISP.

Each solution monitors and stores data in a slightly different way. However, all monitors perform the majority of data analysis on the cluster itself, so data structures are computed locally. The goal of Clusmon is to perform only data collection on the cluster, and move the data analysis to another location. Clusmon is intended to be a design alternative.

1.3 Prior work on Clusmon

Clusmon was declared in-progress long before I became a student at Boise State University. In 2004 an undergraduate named Joey Mazerelli started Clusmon as a

project for a Parallel Computing course under Dr. Amit Jain. His initial goal was "to simply have fun and learn more about clusters and operating systems (and get a good grade in the class)!"

A hard-core web developer, Joey wrote most of the original PHP front-end and designed the beautiful interface graphics. The most complex of these are the dynamic averaging graphs on the dashboard he calls the "Clusometer". Most of the other components are static images that change based on user-defined metrics.

Joey's original design was based on Dr. Jain's wish to have three separate daemons. As was probably the best choice, Joey wrote all three daemons in C. After several months Clusmon was fairly complete and functional, but not very reliable. It couldn't run for more then several weeks without crashing and/or hanging the cluster. This was attributed to memory leaks resulting from void pointers and large, complex data structures. Clusmon was left unusable and unfinished.

I chose to lift Clusmon from the ashes, and create a complete Beowulf cluster monitoring system as a project.

Chapter 2

HIGH-LEVEL DESIGN

2.1 Three-tiered approach

It was decided in the conceptual states that Clusmon would have a back-end, a middle layer, and a front-end. This three-tiered approach separates the gathering, storing, and displaying of cluster data into three distinct steps, and also offloads much of the work from the master node. The collection work will be done by daemons: processes that run in the background.

2.1.1 Back-end

The back-end consists of two daemons: a node daemon and a head daemon. The node daemon runs a copy on each individual worker node gathering data from the operating systems and probing hardware sensors. The head daemon runs on the master node, collecting the data objects from the worker nodes. The head daemon then forwards the collection of data objects to the web daemon.

2.1.2 Middle Layer

The middle-end consists of the web daemon and a database. The web daemon runs on a web server offsite from the cluster *(recommended)* or it can run on the master node. The web daemon receives the collection of data objects and inserts them into a database. The database contains rows of data from each worker node.

2.1.3 Front-end

The front-end is the web interface. The interface, written in a dynamic web language, uses the data from the database to display the data in an easily readable form.

Figure 2.1 shows the design.

2.2 What Data To Gather?

2.2.1 The `/proc` file system

Since Beowulf clusters are Linux-based, kernels on each node will include the `/proc` virtual file system. It's a directory of plain-text files containing the runtime state of the operating system. Figure 2.2 shows an example of `/proc/meminfo` data.

This data can be parsed and displayed to the user as "memory usage".

Other useful `/proc` files include:

- `/cpuinfo` - CPU type, model, MHz, cache size, etc. for each processor
- `/loadavg` - current system load

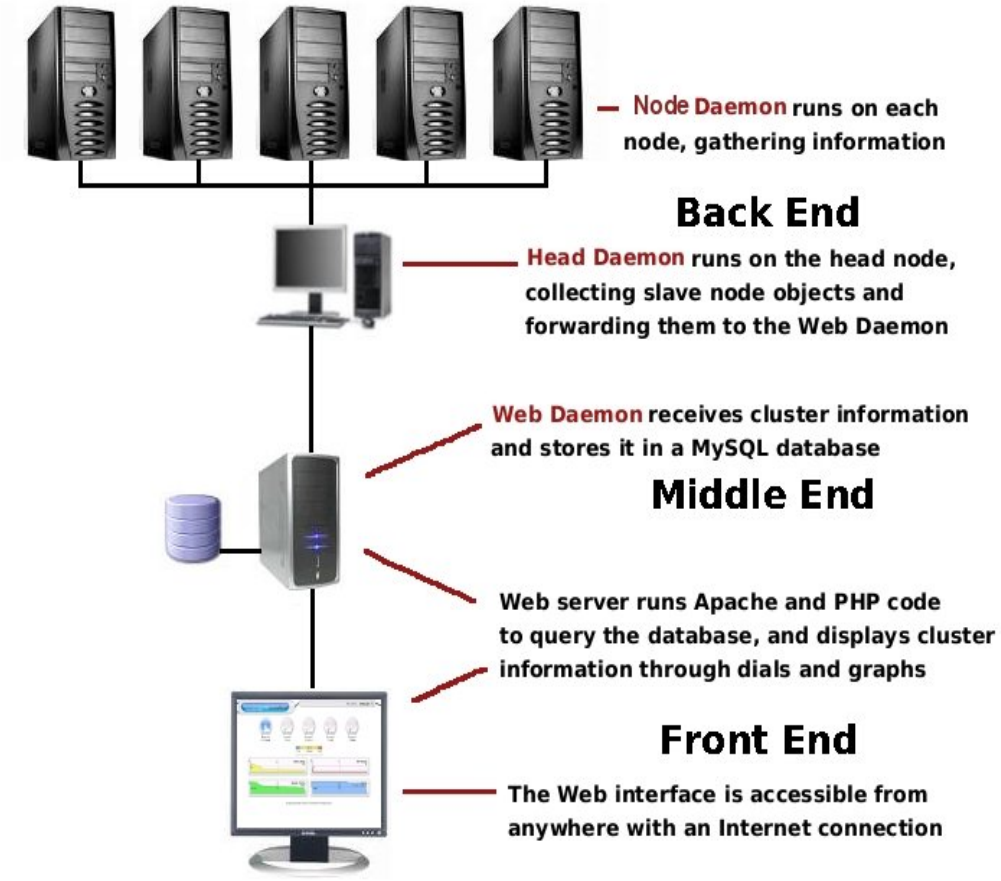

Figure 2.1. Three-tier design

- `/uptime` - length of time since the system was booted
- `/stat` - CPU usage, running processes, etc.
- `/net/dev` - network activity in kilobytes and packets for each network interface

A quick way to parse this data is described in the chapter on optimization.

```
MemTotal:       483648 kB
MemFree:          8308 kB
Buffers:        139208 kB
Cached:          76228 kB
SwapCached:          0 kB
Active:         322312 kB
Inactive:        78736 kB
HighTotal:           0 kB
HighFree:            0 kB
LowTotal:       483648 kB
LowFree:          8308 kB
SwapTotal:      971892 kB
SwapFree:       968672 kB
Dirty:               8 kB
Writeback:           0 kB
```

Figure 2.2. `/proc/meminfo`

2.2.2 lm_sensors

Unfortunately many distributions and kernel versions of Linux don't include a driver for probing temperature sensors. Luckily there's a 3rd-party tool called lm_sensors [6], an open-source development project. lm_sensors is a Linux driver that detects motherboard and CPU sensors, and displays temperature, fan speeds, voltage information, etc. It supports a wide range of hardware, and even includes a library for program integration. Figure 2.3 shows an example of lm_sensor output.

Probing sensor data turns out to be the most time-consuming part of Clusmon—refer to Chapter 7.2.

```
it87-isa-0290
Adapter: ISA adapter
VCore 1:   +1.39 V  (min =  +1.42 V, max =  +1.57 V)   ALARM
% Core 2:   +1.46 V  (min =  +2.40 V, max =  +2.61 V)   ALARM
+3.3V:     +6.66 V  (min =  +3.14 V, max =  +3.46 V)   ALARM
+5V:       +4.92 V  (min =  +4.76 V, max =  +5.24 V)
+12V:     +11.78 V  (min = +11.39 V, max = +12.61 V)
-12V:     -19.87 V  (min = -12.63 V, max = -11.41 V)   ALARM
-5V:       -2.35 V  (min =  -5.26 V, max =  -4.77 V)   ALARM
Stdby:     +4.89 V  (min =  +4.76 V, max =  +5.24 V)
VBat:      +3.25 V
fan1:     1205 RPM  (min =    0 RPM, div = 16)
fan2:        0 RPM  (min =  332 RPM, div = 16)
fan3:        0 RPM  (min =  664 RPM, div = 8)
M/B Temp:    +36C  (low  =   +15C, high =   +40C)   sensor = thermistor
CPU Temp:    +38C  (low  =   +15C, high =   +45C)   sensor = thermistor
Temp3:       +40C  (low  =   +15C, high =   +45C)   sensor = thermistor
```

Figure 2.3. sensors output

Chapter 3

LOW-LEVEL DESIGN

3.1 Design Considerations

As a monitoring system Clusmon runs 24/7. It must be fast, use minimal resourses, and run secondary to the cluster's main purpose: to crunch large amounts of data. Other design considerations include: usage of threads, race conditions, networking, database design, objects, and language choice.

Threads

Rather than rapidly probing the worker nodes, Clusmon only periodically probes the nodes based on a user-defined interval, say every 10 seconds. Programming timers and handling simultaneous tasks requires the use of threads.

Networking

Worker nodes report to the master node periodically via network communication, and the master node forwards the information to the web server. Extensive use of network communication is necessary.

Databases

A MySQL database will store node information. Database updates will need to occur frequently, perhaps in every one to ten seconds. Several tables must be maintained and designed to be compact to facilitate long-term storage.

Objects

There are a lot of potential objects. Each node is represented as an object, for example. As the node state changes, the node object changes. Objects are easily represented and passed around. An object-oriented approach is probably best.

Semaphores

Several list structures are needed, such as a linked-list to house worker node objects, and a linked-list of worker node network connection objects. Linked lists must be protected from race conditions, so only one thread must be allowed to alter the list at a time. Semaphores are needed to provide this protection.

3.2 Picking the Language

When it comes to speed of execution, C is the language of choice. But we want an object-oriented approach, which C doesn't provide. C++ allows objects, but consider the other requirements: semaphores, threads, and networking. C++ can do these things, but they are difficult to maintain. Network code in C++ is esoteric, because

it was added as an after-thought. Threads and semaphores are equally difficult to set up. What we need is a language that's designed for distrubuted computing, a language like Java. As is the case with most distributed systems, Java is a cleaner alternative.

3.2.1 C vs Java

What about speed? Isn't Java much slower than native languages like C? Yes, but speed isn't the issue. The requirements of Clusmon aren't necessarily raw execution speed, but rather efficiency. Does it matter if node information is posted to the database 4ms slower? Probably not. The important thing is that the daemon running on the worker node uses as little resources as possible. A Java back-end brings multiple benefits:

- The code is more readable and maintainable
- Threads are well defined, started, and finished with automatic clean-up
- Garbage collection is automatic
- Networking is built in, and much cleaner
- RMI allows data to be passed as intuitive objects
- Every Java class is a monitor so list synchronization is simple
- JDBC allows an interchangeable interface with databases

- The code is more portable between Linux distributions
- JAR files allow more streamlined distribution

3.3 Is Java fast enough?

Before writing one line of code, we had to be sure that Java would be fast enough for a monitoring system. Fast computations traditionally have been written in C, but Java for the most part has been ignored as too slow to be useful. But is it really that slow? Investigating this question was the first step inthe research.

The stigma that Java is slow just isn't true anymore. In some cases it's just as fast as a native language, depending the constructs used. For example, Linux Networx published a very convincing whitepaper titled *High-Performance Linux Cluster Monitoring Using Java*[7], which demonstrates parsing algorithms, and shows that parsing in Java is comparable to C. Requirements and benchmarks are similar, so it was decided that Java is fast enough for Clusmon.

Chapter 4

DATA COLLECTION

4.1 Node Daemon

The back-end of Clusmon is the part that runs on the cluster itself: the node daemon and the head daemon. The node daemon gathers and parses system sensor data based on settings in the `clusmon.node.config` file. There are two versions of the node daemon: a stand-alone mode and a node mode.

Stand-Alone Mode

Stand alone mode is designed to run on a personal computer and logs its data to the clusmon log directory. It assumes no master node and no web server. To initialize stand alone mode one runs:

```
java NodeDaemon [interval]
```

The interval is the number of seconds between appends to the log file. If the log file doesn't exist, then it's created.

Node Mode

The node mode is the default. It assumes a head daemon is running with the same *unblocked* multicast group IP and port number. Firewalls are common, so one must be sure the multicast IP and ports are open. Starting the node daemon is this mode is typically done using pdsh and a script, see Appendix C.1.

When the node daemon starts up it makes its presence known by sending a multicast. This multicast registers the worker node with the head daemon, but it won't receive an immediate acknowledgment since multicasts use UDP ports. The acknowledgment will come later when the master node calls getNodeInfo(), an RMI method in the node daemon. This method is called every update interval—the default is 10 seconds.

Also in the `clusmon.node.config` file is a timeout setting. This setting tells the node daemon how many seconds to wait for the head daemon to call getNodeInfo(). By default, if getNodeInfo() isn't called within 15 seconds, the node daemon assumes the head daemon is down, and will attempt to re-register itself with the head daemon every 15 seconds, until the master node comes back up.

4.2 Head Daemon

The head daemon is the most complex part of Clusmon. It performs all the RMI calls, contains most of the threads, and maintains connections to each worker node as well as the web server. It's truly the "hub" of Clusmon.

```
// "it" is an iterator of the call list
while(it.hasNext())
{
   current = (WorkerNodeObject)it.next();
   // assign the current RMI call to a thread and start it
   pool.assign(new WorkerThread(current, nodeObjectDataList,
                                nodeConnectionList, interval));
}
```

Figure 4.1. Loop that calls all worker nodes from the master node

When the head daemon starts up, it spawns a thread that indefinitely listens for new worker node multicasts. If the head daemon receives a multicast packet from a node daemon, it adds its connection information to a list of connection objects. This list is used at each update interval.

The most important task for the head daemon is initiating the update cycle. This cycle is regulated by a timer that goes off at an interval set in the clusmon.head.config file. Each cycle starts with calling all of the worker nodes in the call list. These calls are assigned to a thread pool and done simultaneously (Figure 4.1).

Once all data is returned by the worker nodes, the head daemon packages the data, and forwards it to the web daemon. If a node didn't respond to the call, then the head daemon notifies the web daemon via the RMI method setNodeDown(node_id). If the node comes back up, it is simply reintegrated.

Chapter 5

DATA STORAGE

5.1 Web Daemon

When the the web daemon starts up it attempts to connect with a MySQL database. The database can be local or remote, but must be running and have correct login parameters. Once a database connection is made, the web daemon just listens for updates from the head daemon.

5.1.1 Updating Current Statistics

The web daemon receives updates through the RMI method setNode(list, users, history). When this method is called, a thread is spawned and the method returns. This way the head daemon doesn't need to wait until the update is finished. With the newly spawned thread, the web daemon breaks up the list of worker data, and inserts them into prepared statements. A prepared statement is an extremely efficient way of sending batch updates to a database with one connection (Figure 5.1).

Since database updates always involve multiple nodes, the web daemon has prepared statement like this throughout the code. The web daemon loops through the

```
PreparedStatement p = conn.prepareStatement("UPDATE nodes_info SET " +
"node_id=?, node_name=?, node_ip=?, cpu_count=?, stopped_fan_mask=?, " +
"fan_count=?, total_memory=?, uptime=?, fan1_rpm=?, fan2_rpm=?, fan3_rpm=?," +
" fan4_rpm=?, fan5_rpm=?, fan6_rpm=? WHERE node_id =?;");
```

Figure 5.1. Prepared Statement for a MySQL database in Java

list adding batches on the fly. Once all batches have been added, the updates are sent to the database in one statement (Figure 5.2).

5.1.2 Storing Historical Data

The web daemon updates all tables at the update interval, except one: the `past_stats` table. This table isn't current information, but rather a historical archive. `past_stats` is updated at an update interval multiple defined in the clusmon.head.config file. For example, if the update interval is 10, and the multiple is set to 6, then the `past_stats` table is added to every 60 seconds, or every 6th time the interval is reached. In the setNode(list, users, history) method, the *history* argument is a boolean. When this argument is set to *true*, the web daemon averages all the current information, and dumps it into `past_stats`.

Since `past_stats` grows indefinitely, it's important to understand how much space one record takes.

One record takes 38 bytes. If one record is inserted once per minute: 38 bytes $*$ 60 minutes $*$ 24 hours $*$ 365 days $=$ $19,972,800$ bytes. That's approximately 20 megabytes per year—negligible by today's standards. The default location for Clus-

```
for(int i = 0; i < list.size(); ++i)
{
   NodeData current = (NodeData)list.get(i);
   if(!insertedNodes.containsKey(new Integer(current.node_id)))
   {
      p.setInt(1, current.node_id);
      p.setString(2, current.myName);
      p.setString(3, current.ipAddress);
      p.setInt(4, current.cpu_count);
      p.setInt(5, current.stopped_fan_mask);
      p.setInt(6, current.num_fans);
      p.setInt(7, current.total_mem);
      p.setInt(8, current.uptime);
      // try using the value in the array, otherwise set to 0
      try { p.setInt(9, current.fan_speeds[0]); }
      catch(Exception e){p.setInt(9, 0);}
      try { p.setInt(10, current.fan_speeds[1]); }
      catch(Exception e){p.setInt(10, 0);}
      try { p.setInt(11, current.fan_speeds[2]); }
      catch(Exception e){p.setInt(11, 0);}
      try { p.setInt(12, current.fan_speeds[3]); }
      catch(Exception e){p.setInt(12, 0);}
      try { p.setInt(13, current.fan_speeds[4]); }
      catch(Exception e){p.setInt(13, 0);}
      try { p.setInt(14, current.fan_speeds[5]); }
      catch(Exception e){p.setInt(14, 0);}
      p.setInt(15, current.node_id);
      p.addBatch();
   }
}
p.executeBatch();
```

Figure 5.2. Loop that inserts object data into a prepared statement

TABLE 5.1 Record size in past_stats

Field	**Type**	**Bytes**
date_time	datestamp	8
avg_temp	int	4
low_temp	int	4
high_temp	int	4
mem_avg	int	4
cpu_avg	int	4
network_avg	int	4
uptime_avg	int	4
nodes_used	smallint	2
	Total	38

mon database files is `/var/lib/mysql/clusmon_data`

5.2 Clusmon Database

Clusmon uses a MySQL database to store current and historical data. The clusmon.web.config file contains database connection inforation used by the web daemon. The database can reside locally with the web daemon, or on a remote machine. See Appendix F for table descriptions.

For connectivity, Clusmon uses the JDBC (Java Database Connectivity) interface. JDBC is an API that allows interchangeable database drivers, and provides methods for querying and updating data in a relational database. The JDBC driver and appropriate permissions given to the web daemon are required for everything to work correctly.

Chapter 6

WEB INTERFACE

6.1 Web Interface

The web interface is written in PHP. It's a good web language for dynamic graphics and connecting with a MySQL database. The interface is very flexible because it gets all of its data from the MySQL database, so the entire page design can be altered at any time without stopping Clusmon.

6.1.1 clusmon.php

The file `clusmon.php` in Figure 6.1 is considered the main page of Clusmon. It represents a snap-shot of the cluster as a whole. The five circular gauges represent percent usage in a color scheme consistent throughout the interface. Blue means little or no usage, green is low, yellow is medium, red is high, and black is down. It also contains four dynamic graphs with various usage information. If temperature data is gathered, then the low, average, and high temperatures are displayed as well. Two buttons in the upper right labeled "past stats" and "node list" navigate to their respective pages.

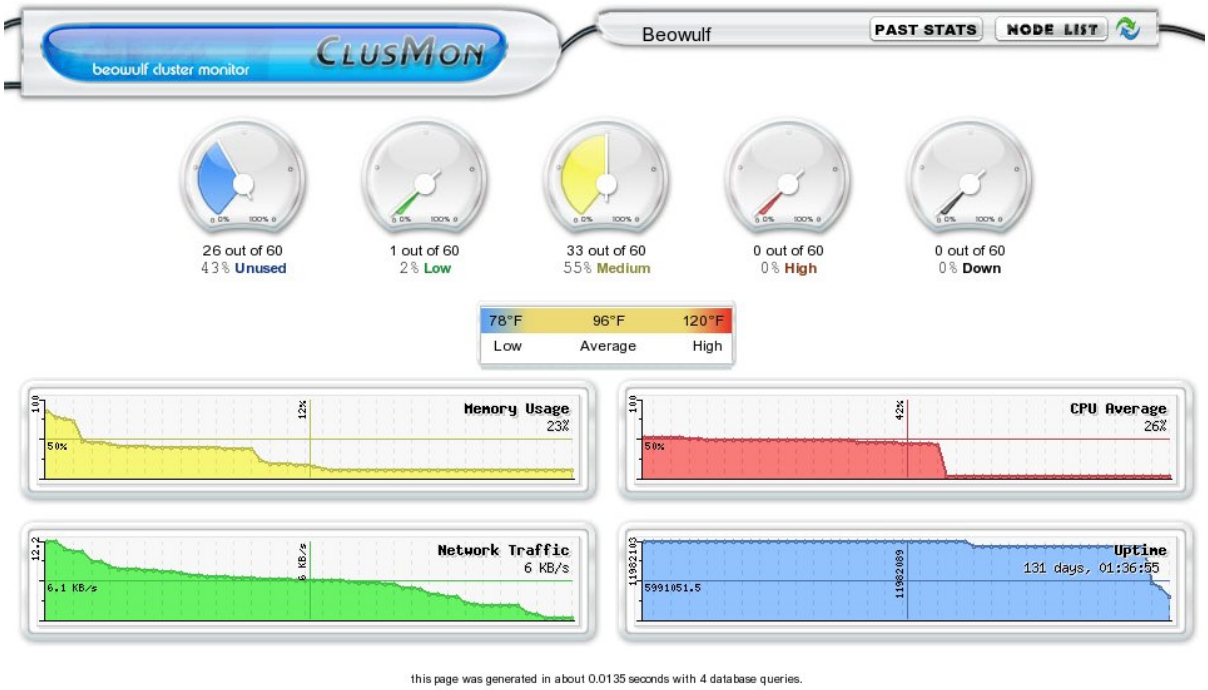

Figure 6.1. `clusmon.php`

The Refresh Button

The grapic seen in Figure 6.2 resembling two arrows is an auto-refresh button. When activated it sets the page to automatically refresh every `n` seconds. This gives the page a "real time" effect. If the page is no longer in the foreground, (the user clicks off the page, or opens another tab,) auto-refresh is paused. There's no reason to refresh the page if the user isn't looking at it. When the page comes back into the foreground, auto-refreshing is resumed. The `history.php` page doesn't auto-refresh, however, because graphs take a long time to load. The refresh interval is set in `class.settings.php`; and cookies must be enabled in the web browser to use this

feature.

Figure 6.2. Auto-refresh button.

6.1.2 nodes.php

The file `nodes.php` in Figure 6.3 shows a list of individual node information sorted by node id as default. This page is useful to monitor nodes at an individual level, to see which nodes are most heavily used, and pinpoint problems. At the top of each column is a button that describes the column. When clicked, these buttons sort the list associated with the column. For example, you can sort by temperature, or CPU usage. If the same button is clicked again, the sorting alternates from descending to ascending order. Node names are also links which open a new small window to display more information (Figure 6.4). This display may not match the information in the list because it's another query to get the most up-to-date information.

6.1.3 history.php

The file `history.php` in Figure 6.5 shows data gathered over time. It contains a drop-down list of historical graphs, radio buttons, and text boxes to create a time range. When "draw" is clicked, it generates a line graph based on the time range. The wider the time range, the longer it takes to generate. This is because it draws a

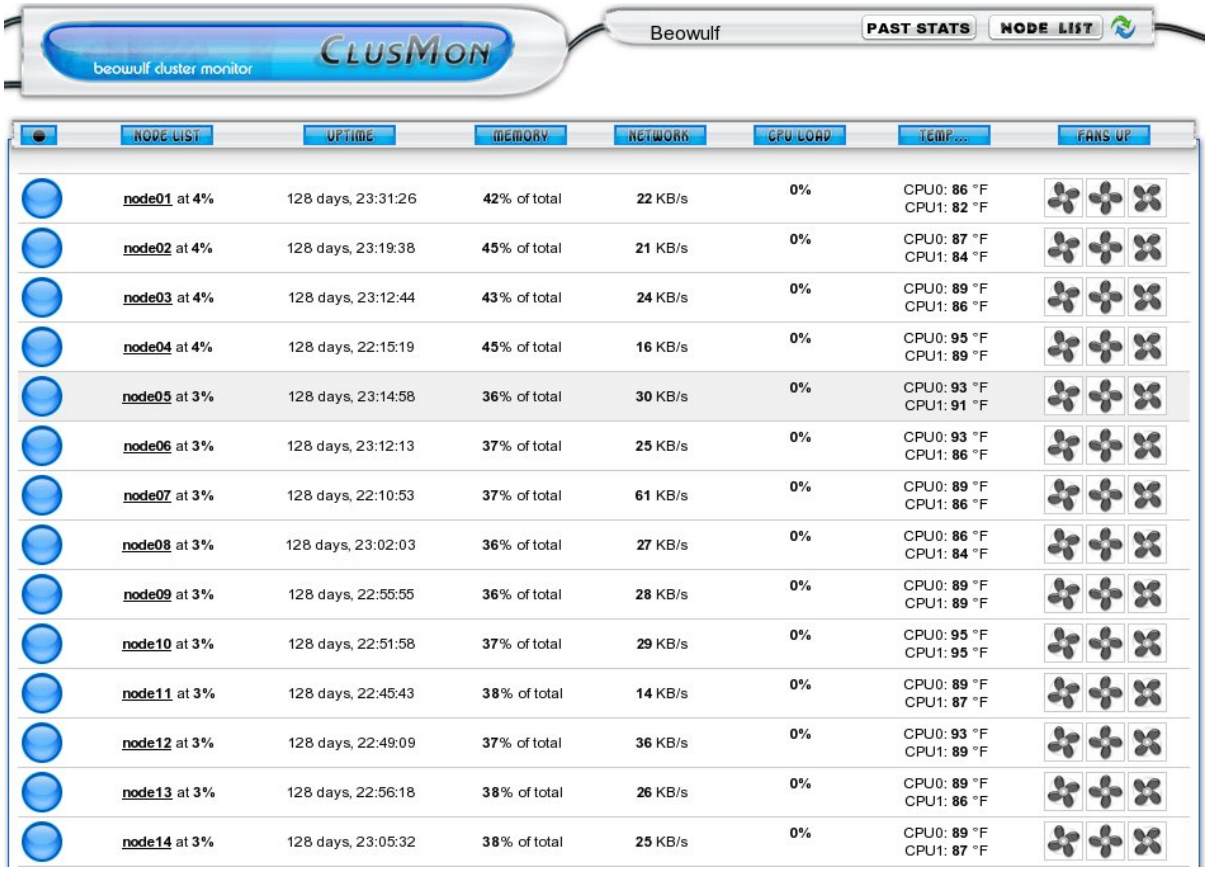

Figure 6.3. `nodes.php`

line for each pair of points; and one month could contain over 50,000 points!

The drop down list allows multiple selections. If the user holds the CTRL button while clicking multiple graphs, they will be selected and each of them displayed. This is particularly useful when comparing two graphs (Figure 6.6). This example shows how average temperature is effected by CPU usage. Most graphs have some type of correlation—CPU, memory, network traffic, nodes used, and temperature will generally go up and down together.

These particular graphs are generated using a 3rd party package called JPGraph [8],

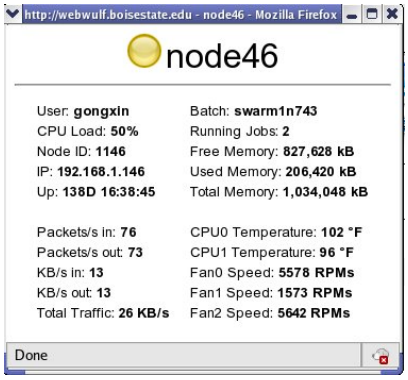

Figure 6.4. `nodedetail.php`

an open source graph and chart generator. JPGraph was developed under the QPL license. It can be freely used privatly and commercially, but it cannot be sold without consent. If you would like the license of Clusmon to change to GPL, eliminate any references to JPGraph, which can easily be done by eliminating the `history.php` page.

As an important note: by default php.ini allows an 8MB max of post data. This is not enough for most graphs! It is recommended that the limit be increased, to 16MB, or even 64MB, or more! If no graph is generated, there's a good chance this number is too low.

6.2 gd_library

The gd_library was written in C and is used for dynamic image creation. All dynamic graphs in Clusmon are PNG images created on the fly using the gd_library; so it is required. If you are unsure if you have GD library, you can run `phpinfo()` to check

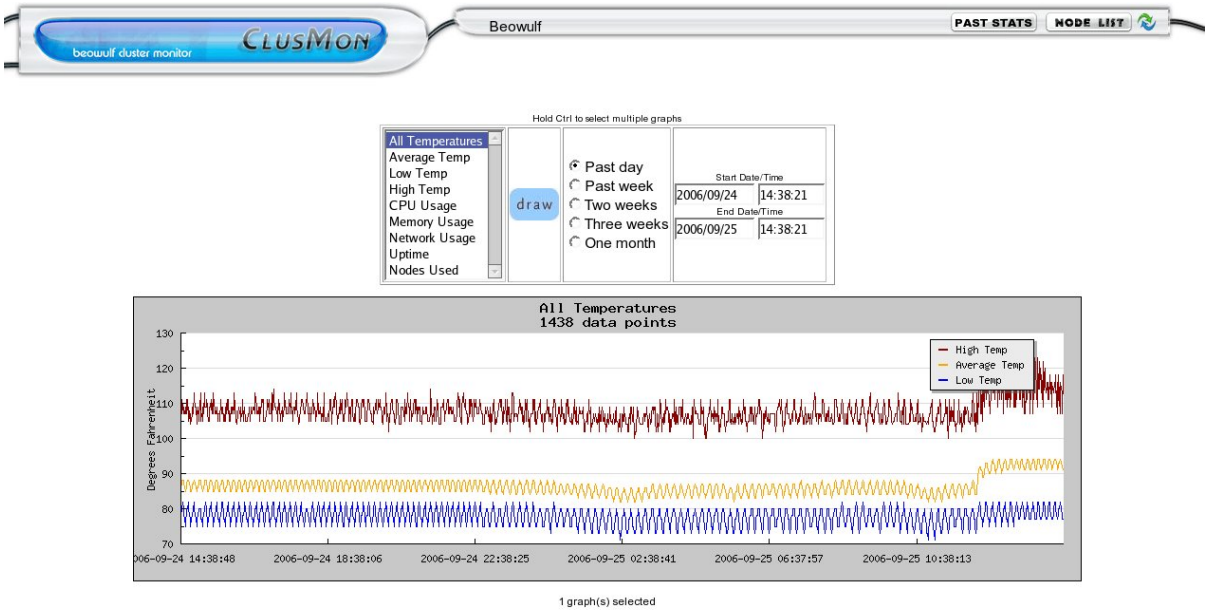

Figure 6.5. `history.php`

that GD Support is enabled. If you don't have it, you can download it for free [9].

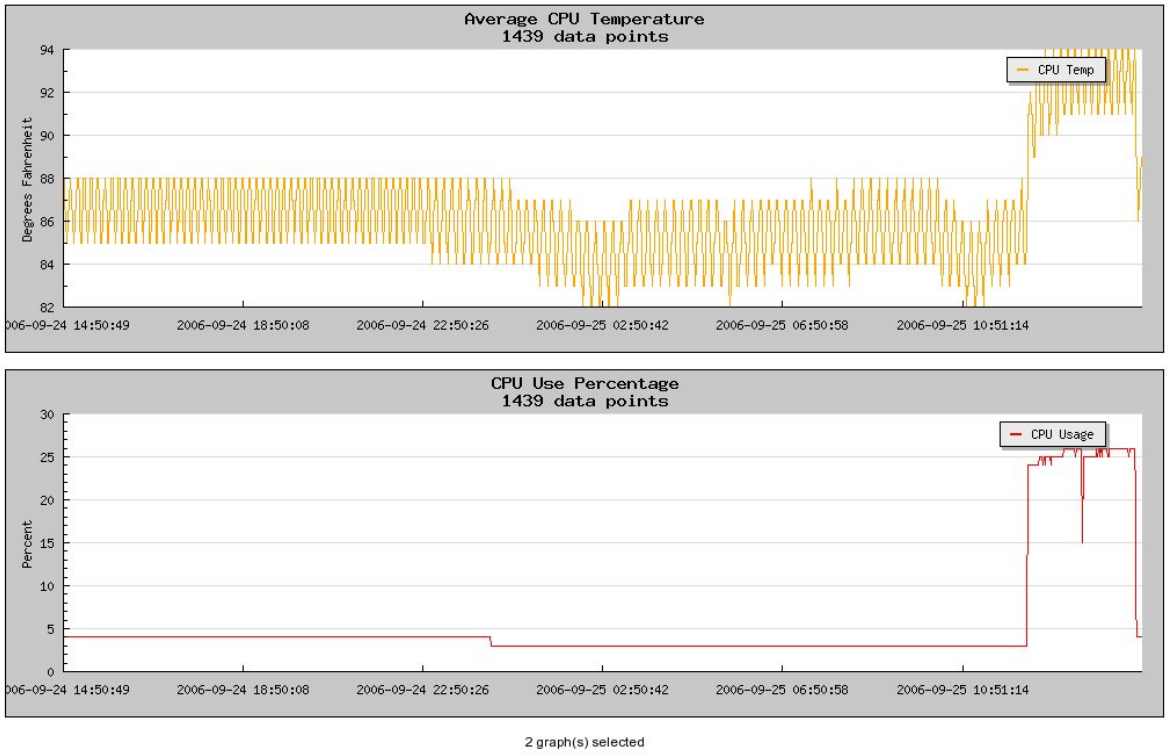

Figure 6.6. Comparing two graphs

Chapter 7

OPTIMIZATION

7.1 Finding the Bottlenecks

From language choice to data structures, optimization was considered at every stage during Clusmon development. But optimization was done with the old saying in mind, "pennies on the dollar." This applies to the informal rule that 90% of the time is spent in 10% of the code. In order to optimize, we need to find where all the time is being spent.

7.2 Performance Breakdown

The most oft-repeated event in Clusmon is the update cycle. It begins when the head daemon makes its first call to a worker node, and ends when the last node is updated in the database. Much of this cycle time depends on option settings in the config files.

The following data were obtained on Boise State Beowulf cluster (See Appendix D.1.1). With all options turned on, and a pool of 60 threads, here are the average times (with events occurring in chronological order):

TABLE 7.1 Performance breakdown of one update cycle

Task Time	**Daemon**	**Task**
3ms	Head	RMI request to each node (split into 60 threads)
1100ms-1600ms	Worker	Querying and parsing lm_sensors
2ms	Worker	Collecting and parsing memory data
1ms	Worker	Collecting and parsing network data
30ms	Head	qstat call and parsing batch data
70ms	Head	Pushing all cluster data to the web daemon
5ms-8ms	Web	Creating a database connection
40ms-45ms	Web	Pushing updates to the database
1250ms-1800ms		

Obviously the lm_sensors take most of the time, about 90%. If lm_sensors are turned off in the config file the results are much faster. Total time drops from an average of 1500ms down to a mere 200ms.

This begs the question, are lm_sensors worth it? Ninety percent of the time to gather ten percent of the data? Is the performance hit worth temperature and fan data? Yes, it's worth it—because the lm_sensor driver query is IDLE TIME. It's a round-trip to the sensors on the chip, not a busy wait. That extra second to gather lm_sensor data is just extra time, not resources. It takes about a second for most web pages to load anyway, so using lm_sensors makes little difference. Regardless, lm_sensors is where most of the time is spent, so optimization efforts should be focused here.

7.3 Optimizing Java

No stone was left unturned with regard to an efficient use of resources and execution speed. This section describes investigations on optimization.

7.3.1 Java Virtual Machine

The Java Virtual Machine is what allows Java to "write once, run anywhere". It's the layer between Java class files and the native binary, acting as an interpreter. The JVM is required to run on any computer executing Java code.

The JVM has long been regarded as the crux of Java performance. In Clusmon, it's not the extra layer that poses the problem, but the resources to run the JVM. From growing and shrinking the heap to garbage collection, resource allocation is automatic, and the JVM will take all the memory it needs.

The JVM does use resources, but is actually quite efficient; and there is a way to minimize its impact. The command **-mx** sets the maximum heap size the JVM is allowed to use. Clusmon sets the limit to 10 megabytes **-mx10m**, giving Clusmon a smaller footprint. It frees up that much more memory for other processes. In this case, the resources used by the JVM are worth the tradeoff.

7.3.2 GNU Java Compiler

It is possible to execute Java programs without the JVM. The GJC [10] is a limited compiler that compiles Java code into native Linux bytecode. This would not only eliminate the need for the JVM, but theoretically speed program execution because the code becomes native. However, *limited* is emphasized because the GJC is too limited for Clusmon. RMI isn't supported, among other things; so this avenue was quickly abandoned.

7.3.3 lm_sensors and JNI

Obviously, lm_sensors is the slowest part of Clusmon. Most of the time is spend in the sensor query, and the rest is spent in parsing sensor output. lm_sensors is open source, but attempting to optimize by tampering with the complex C driver was beyond the scope of the project. However, lm_sensors documents a library that allows direct calls from C code. A direct call would elminiate expensive console output parsing. But Clusmon was written in Java, which can't directly call C. Fortunately, there's the Java Native Interface

The Java Native Interface is a programming framework that allows Java code running in the Java virtual machine to call and be called by native applications ... and libraries written in other languages, such as C, C++ and assembly [11]. Basically, it allows Clusmon to directly call the lm_sensor code. With some cryptic coding this actually worked, and saved about 200ms. However, problems surfaced. Clusmon gobbled up 1 full megabyte every 5 minutes, and never released it. As it turns out, lm_sensors has memory leaks! To be sure, I used Valgrind to verify the JNI code. It traced all un-released memory to the lm_sensors calls. These memory leaks are documented, and the Java garbage collector doesn't clean them up.

A possible way around this was to unload the lm_sensor shared library every so often at runtime. Theoretically, this would release the memory. Unfortunately, just such an attempt disproved the theory. The memory wasn't released.

And so Clusmon is stuck with parsing the console output, which isn't a such bad

thing. Even if using the JNI did work, it wouldn't have addressed the real problem: the round-trip to the sensors.

7.4 Java Networking using RMI

Clusmon uses RMI instead of TCP sockets. RMI (Remote Method Invocation) allows Java objects to invoke methods running on remote machine [12]. It looks just like a method call. Clusmon's head daemon makes an RMI method call to each of the worker nodes, and objects are returned. Once all the objects are returned, they're put into a linked list, and another RMI call pushes the list to the web daemon to be pushed into the database.

Why does Clusmon use RMI? A comparison of RMI vs TCP below shows that RMI is the better choice for this particular application.

7.4.1 RMI vs TCP

It's awkward to compare RMI and TCP, because RMI uses TCP under the hood. A side by side comparison shows RMI transmits more packets because the additional method data overhead. But it doesn't mean TCP is always faster. It all depends on implementation.

In RMI, the connection is always open, so connection overhead is very low. TCP, on the other hand, usually closes and re-opens connections—depending, of course, on programmer implementation. TCP is optimal with short bursts or continuous streams

of data. However, RMI shines when the payload is large and relatively infrequent, such as Clusmon updates. Because of the large payload, the additional overhead of RMI is negligible. Not to mention RMI has the advantage of having more maintainable code.

7.5 Java Threads

Threads are used extensively throughout Clusmon. They are necessary to make things happen simultaneously, and account for most of the measurable performance boosts.

7.5.1 Listener Threads

Clusmon has several "listener" threads that need to always be listening. The head daemon has a listener thread that continuously listens for new worker nodes. If it receives a multicast from a worker node, it adds the worker node's IP address to the RMI call list. Another listener thread is on the worker node. It listens for the master node to request data before a timeout period. If a timeout occurs, the node daemon attempts to reconnect with the head daemon by sending another multicast.

7.5.2 Timer Threads

Perhaps the most important thread is the interval timer. This thread schedules updates to occur every n seconds. If the web daemon goes down, however, the timer thread is killed. It is restarted when the web daemon comes back up.

7.5.3 Performance Threads

`"...and in haste return." -Beowulf`

Originally, the master node incrementally called each of the worker nodes, waiting for a reply before calling the next one. This was *very* slow and inefficient. To speed things up, Clusmon spawns a thread for each RMI call making them occur simultaneously. Since spawning a thread incurs overhead, Clusmon uses a thread pool. The threads in the pool are initialized only once, and always ready to run. Now all of the calls return almost simultaneously, (requiring a lot of synchronization!). But the performance gain is phenomenal: about number-of-nodes times faster. The `clusmon.head.config` contains a field with the number of threads in the pool.

Another performance thread is on the web daemon. When the head daemon calls the method that pushes the data to the web daemon, it returns immediately because the web daemon spawns a thread that performs database inserts.

7.6 Parsing the Data

Analyzing and separating text into component parts, also known as parsing, is an important part of Clusmon. It's the only way to copy text values into variables. There are several options for effective parsing, namely tokenizing and regular expressions. Clusmon uses both.

7.6.1 Tokenizing

Peculiar to Java is the StringTokenizer class. It separates strings with delimeters, and is very useful when the data format will not change, such as in some `/proc` files. Below is an example of a line of code in NodeDaemon.java that expects the next token to contain a string that can be converted to a float.

```
data.uptime = (int)Float.parseFloat(uptime_token.nextToken());
```

The StringTokenizer class is a quick and dirty parsing technique. However, if the data format changes, it breaks.

7.6.2 Regular Expressions

In the cases where data format changes frequently and tokenizing isn't enough, Clusmon uses regular expressions. Regular expressions are widely used in many programming languages. It's essentially a language used for describing and matching string patterns. Regular expressions are more expensive then tokenizing, but are much more powerful. Below is an example in NodeDaemon.java that attempts to match any number 0-9 followed by exactly one character of anything else.

```
Pattern value = Pattern.compile("[0-9]+");
```

Whether tokenizing or using regular expressions, parsing is a critical point in Clusmon because it determines compatibility across Linux distributions. It's "where the rubber meets the road" in data collection. And since it's done so often, it should

be optimized. Attention was given to parse optimization, even though most of the execution time is spent in file I/O and network communication. More could be done; for example: Java could use ASCII instead of Unicode when reading in text files, but it doesn't go this far because parsing isn't a bottleneck in Clusmon.

Chapter 8

POST-DEVELOPMENT REVIEW

8.1 Clusmon in Action

When I got the code to a point where I could run Clusmon on the Beowulf cluster, the data returned was immediately useful. The first thing I noticed was a bad system fan. The list of nodes showed node21 with a fan running at 0 RPMS. It turns out the fan was running, but the sensor was bad. Clusmon pinpointed a potential problem within minutes.

In another instance Clusmon caught a problem with the CPU usage on an individual node. A job running on node52 had 100 percent CPU usage, but the same job on other nodes only showed 50 percent usage. The anomaly was caused by CPU averages in `/proc`. All nodes were configured for hyperthreading except node52, causing `/proc` averages to be off by a factor of 2. Node52 was somehow missed several months earlier while the cluster was being configured for hyperthreading, so it wasn't running to its full potential. We fixed that in a hurry.

Perhaps most impressive was when Clusmon caught a problem with the Cluster as a whole. One evening while I was coding the interface, I noticed the average

cluster temperature starting climbing—90 degrees, 100 degrees, 110 degrees. This was abnormally high, so I send a concerned emailed to Dr. Jain. He immediately went down to the lab and discovered the air-conditioner had kicked off. He turned the air-conditioner back on, and stayed until Cluster temperatures went back to normal. This spawned a new Clusmon feature. Now Clusmon automatically sends a warning email to the cluster administrator if the average temperature ever exceeds the threshold set in the `clusomn.head.config` file.

8.2 Clusmon Limitations

Though Clusmon is very useful, it has its limitations. Clusmon isn't designed to run excessively frequent updates. It's not going to say what the cluster is doing every millisecond. In fact, the shortest interval possible is one second. Any faster doesn't make much sense because most files in `/proc` won't provide new information that fast anyway. Additionally, the interface is a web page—hardly suitable for administrators wanting to know what's going on this very instant. Typically, a ten to thirty second interval is adequate.

Another obvious limitation is a loaded network. If the cluster network is heavily loaded, Clusmon may temporarily fail to function because there's no remaining bandwidth to pass Clusmon data. However, this isn't the same as a heavily loaded CPU, where Clusmon performs just fine. The network issue can be resolved by having an alternate admin network, as some clusters have done. With two network cards,

Clusmon can run on the network not used by parallel processes.

8.3 Java Quirks

Coding Clusmon in Java went very smoothly. There was never a language barrier that prevented me from adding any features. However, there were some quirks that required creative work-arounds. For example, the three Clusmon daemons had to be able to start in any order, die, and restart gracefully. This meant putting threads on hold, saving state, and re-establishing network connections.

The problem came when I tried to pause a timer thread. The head daemon uses a built-in Java method called scheduleAtFixedRate(TimerTask task, long delay, long period). It's a very handy method for scheduling tasks to run at a set interval. If the web daemon goes down, there's no need for the head daemon to gather info, so I attempted to "pause" the scheduler. This worked fine, until I resumed the scheduler. Unbeknownst to me, the scheduler not only resumes, but goes ballistic trying to "catch-up" on missed time. As a side-effect, the nodes are probed in rapid succession until the scheduler has caught up. I couldn't pause the scheduler, I had to kill it! So, the scheduler is killed, and restarted when the web deamon comes back up. Not ideal, but necessary.

8.4 Testing and Debugging

Testing a cluster application is fiendishly difficult. There are more possible problems because there are more computers, and very few clusters to test with. Clusmon was tested on three clusters, and despite the hurdles, it worked quite well with each of them. Parsing was the only challenge, because new data formats would come up. It usually meant I had to change a tokenizer to a regular expression.

Debugging Clusmon was also difficult because there are threads all over the place. Synchronization was a huge issue because sixty threads could simultaneously attempt to add to a linked list. And another thread could attempt to add to a list in the middle of an iteration. Luckily, it's easy to synchronize in Java, so debugging race conditions didn't take long. Java to the rescue!

Chapter 9

CONCLUSIONS

Approximately 400 hours of development went into Clusmon. This doesn't account for the countless hours of testing. Faculty and students both found Clusmon useful during development, so I had a steady flow of bug lists and feature requests. Clusmon even became a permanant fixture on an admin's dual-monitor.

Java helped make Clusmon very stable. It's definitely fast, efficient and maintainable. The unique three-tier design worked better than expected. The middle layer makes it scalable and robust, because the master node isn't bogged down as the number of nodes increase. And the interface can easily be customized. Clusmon is a useful tool for any cluster administrator.

REFERENCES

[1] Interview with Donald Becker. "Beowulf founder: Linux is ready for high-performance computing" by Jan Stafford. SearchOpenSource.com.

[2] Clusterprobe. http://www.srg.cs.hku.hk/srg/html/cprobe/readme_of_clusterprobe.htm

[3] ClusterWorx. http://www.linuxnetworx.com/clusterworx

[4] Ganglia. http://ganglia.sourceforge.net

[5] Supermon. http://supermon.sourceforge.net

[6] Lm-Sensors - Linux Hardware Monitoring. http://www.lm-sensors.org

[7] Curtis Smith and David Henry. "High Performance Cluster Monitoring Using Java". www.linuxclustersinstitute.org/Linux-HPC-Revolution/Archive/PDF02/32-Smith_C.pdf

[8] JpGraph PHP graphing software. http://www.aditus.nu/jpgraph/

[9] GD Graphics Library. http://www.boutell.com/gd

[10] The GNU Java Compiler for the Java Programming Language. http://gcc.gnu.org/java

[11] Java Native Interface. http://java.sun.com/j2se/1.4.2/docs/guide/jni

[12] Java Remote Method Invocation. http://java.sun.com/products/jdk/rmi

Appendix A

PROJECT MANAGEMENT

A.1 Licensing

Clusmon was developed under the GPL General Public Licence version 2 and is available for download from Boise State University. JPGraph was developed by Johan Persson and is under the limited QPL license.

A.2 Where to get Clusmon

The Clusmon home page and a link to download the the code is at `http://onyx.boisestate.edu/clusmon/`. Future project development will be managed by Amit Jain at Boise State University.

A.3 Live Clusmon Clusters

Clusmon is currently running on two clusters at Boise State University. The web sites are `http://webwulf.boisestate.edu/clusmon/` and `http://onyx.boisestate.edu/clusmon/clusmon.php`.

Appendix B

REQUIREMENTS

B.1 Installation requirements

The following is a list of prerequisites for installing and configuring Clusmon.

- Linux kernel version 2.6 with root access
- Java 1.5+
- Apache Server 2.0+
- PHP 5.0+ for the interface
- MySQL Server
- MySQL JDBC driver for interfacing Java with the database
- LM Sensors for tempurature and fan data
- GD Graphics Library for the interface graphs
- pdsh to use the qstat functionality

Appendix C

INSTALLATION

C.1 Installation Guide

Clusmon installation is relatively streamlined; but there are a lot of steps involved, and they must be done in the right order. So plan on spending a good hour. This checklist assumes an administrator-level knowledge of Linux and root access.

1. Decide right now if the master node on the cluster is also going to be the Clusmon web server. This is not recommended because it will put more work on the master node, but it is allowed. If you are setting up a dedicated web server that is *not* part of the cluster, steps 2-4 will need to be repeated for the cluster *and* the web server.

   Copying the files

2. Create a clusmon user with the home directory `/usr/local/clusmon` so it is shared by all nodes across the cluster. This user should be given limited root permissions to start and stop the Java daemons.

   ```
   $ useradd -m -d /usr/local/clusmon clusmon
   ```

3. Download and unpack the Clusmon tarball in the recently created directory. Clusmon is availible at http://onyx.boisestate.edu/clusmon

   ```
   $ tar -xfvz clusmon.tar
   ```

4. Create a `/var/spool/clusmon` directory, and give the clusmon user read/write permissions. This directory is for the clusmon log files.

   ```
   $ mkdir /var/spool/clusmon
   $ chown clusmon:clusmon
   ```

5. Copy the `cluster.node.config` file into the `/var/spool/clusmon` directory. Each worker node will read in their own copy of this file. The log files will be created automatically.

Setting up the MySQL Database

6. Now that all the files are in place, it's time to set up the MySQL database. The database can be installed anywhere, but it's highly recommended that it be installed on the web server. Install MySQL and enable the `mysqld` daemon. Make sure a MySQL root user/password is created. These commands work on Redhat and Fedora; they may be slightly different elsewhere.

```
$ /sbin/chkconfig mysqld on
$ /sbin/service mysqld start
```

7. Create a MySQL clusmon user/password. This account will be used by the Clusmon web daemon. The user/password will need to be set in the `clusmon.web.config` file in a later step.

```
$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is xxx to server version: 5.0.18

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> grant all privileges on *.* to clusmon@localhost identified by
       <password> with grant option;
Query OK, 0 rows affected (0.11 sec)

mysql> quit
```

8. Run Clusmon the `setup.tables.sql` script to create the clusmon database and data tables. Running diff checks to see if the tables were created correctly.

```
$ mysql -u clusmon -p < setup.tables.sql > setup.tables.sql.log
$ diff setup.tables.sql.log clusmon_data_describe
```

9. Setup MySQL Java drivers. In order for Java to interface with the database, the jar file needs to be in the right place.

```
$ cp mysql-connector-java-3.1.8-bin.jar\
 /usr/local/javas/jdk1.5.0_03/jre/lib/ext/mysql-connector-java-3.1.8-bin.jar
```

Setting up the Apache Web Server

10. Now that the database is running, the next step is to set up the Apache web server. Install, configure, and enable the Apache server daemon. Apache needs to be able to run php scripts.

```
$ /sbin/chkconfig httpd on
$ /sbin/config httpd start
```

11. Create a Clusmon html directory to put all of the html and php files.

```
$ mkdir /var/www/html/clusmon
```

12. Copy all the html and php files to the newly created directory. Change ownership of the entire directory so it's public enough to allow access from a browser.

```
$ cp -a /usr/local/clusmon/* /var/www/html/clusmon/
$ chown -R clusmon:clusmon /var/www/html/clusmon/
```

Configuring the Clusmon config files

13. There are three clusmon config files: `clusmon.web.config`, `clusmon.head.config`, and `clusmon.node.config`, one for each daemon. The first two are in the `/usr/local/clusmon/daemons` directory, and the last one should be resident on each worker node in the `/var/spool/clusmon` directory. These config files contain settings that will be loaded each time a daemon is started. If the format wrong or a setting is missing, the daemon will not start. The most important settings in these files are the IP addresses and port numbers. Each worker node has its own config file beacause sometimes different sensors have different formats. Correct settings are very crucial, and have the most potential for problems. Refer to Appendix E and alter the setting in your configuration files as needed.

Configuring the Clusmon daemon scripts

14. The Clusmon daemons are started with scripts that contain path information. They are likely fine as is, but go to the `/usr/local/clusmon/daemons` directory and check the contents of the `run_WebDaemon.sh`, `run_HeadDaemon.sh`, `run_NodeDaemon.sh` to make sure the Java directory is correct.

Starting the Clusmon Daemons

15. With the config files set correctly, it's time to to fire up the daemons. It doesn't matter which order they are started, but lets start with the web daemon. Change to the clusmon user, go to the `/usr/local/clusmon/daemons` directory and start the web daemon.

```
$ sh clusmon_web.sh start
Starting clusmon_web daemon: [  OK  ]
$
```

16. Check the `web.log` file to make sure it started correctly.

```
$ cat /var/spool/clusmon/web.log
09/30/2006 07:45:44--->Starting Clusmon Web Server...truncating tables
09/30/2006 07:45:44--->Listening for updates...
$
```

17. If the Clusmon web daemon started correctly, start the head daemon.

```
$ sh clusmon_head.sh start
Starting clusmon_head daemon: [  OK  ]
$
```

18. Check the `head.log` file to make sure it started correctly.

```
$ cat /var/spool/clusmon/head.log
09/30/2006 07:45:48--->Starting Clusmon Head Daemon...
09/30/2006 07:45:48--->Clusmon Web Server Found! Starting...
$
```

19. If the Clusmon head daemon started correctly, start all the node daemons. This example uses pdsh to start them all with one command.

```
clusmon]$ pdsh -a -x node00 "cd /usr/local/clusmon/daemons;\
 clusmon_node start"
node01: Starting clusmon_node daemon: [  OK  ]
node03: Starting clusmon_node daemon: [  OK  ]
node08: Starting clusmon_node daemon: [  OK  ]
node22: Starting clusmon_node daemon: [  OK  ]
node13: Starting clusmon_node daemon: [  OK  ]
...
clusmon]$
```

20. SSH into one of the individual nodes to make sure it started correctly.

```
$ ssh node01
$ cat /var/spool/clusmon/node.log
09/30/2006 07:46:43---> Sending NEWNODE multicast...
$
```

21. Now go back to the master node, and make sure the head daemon received all the multicasts.

```
clusmon]$ cat /var/spool/clusmon/head.log
09/30/2006 07:45:48--->Starting Clusmon Head Daemon...
09/30/2006 07:45:48--->Clusmon Web Server Found! ...
09/30/2006 07:47:18--->Adding node 113 to the call list
09/30/2006 07:47:18--->Adding node 130 to the call list
09/30/2006 07:47:18--->Adding node 114 to the call list
09/30/2006 07:47:18--->Adding node 129 to the call list
09/30/2006 07:47:18--->Adding node 115 to the call list
09/30/2006 07:47:18--->Adding node 104 to the call list
...
clusmon]$
```

22. Finally, go back to the web server, and make sure the web daemon received data from the head daemon.

```
clusmon]$ cat /var/spool/clusmon/web.log
09/30/2006 07:45:44--->Starting Clusmon Web Server...truncating tables
09/30/2006 07:48:02--->Listening for updates...
09/30/2006 07:48:22--->Number of nodes grew from 0 to 60
clusmon]$
```

23. Congratulations, you have finished installing Clusmon! Open a browser and browse to your clusmon web server and check all the data for accuracy. Happy monitoring!

Appendix D

BUILD AND TEST ENVIRONMENT

D.1 Target Clusters

The following Beowulf clusters at Boise State University were the clusters used develop to test Clusmon.

D.1.1 Beowulf

Beowulf is the main research cluster for computer science at Boise State University with a dual-processor master node and 60 dual-processor nodes.

- private gigabit ethernet
- master node: dual 2.4GHz Intel Xeon processors, 4GB RAM
- other nodes: dual 2.4GHz Intel Xeon processors, 1GB RAM
- Fedora Core 5
- Funded by NFS Grant 0321233.

D.1.2 Rookery

Rookery is an 8-node student research cluster composed of assorted hardware.

- private gigabit ethernet
- master node: 1.7GHz Intel Celeron, 1GB RAM
- nodes 1-3: 1.7GHz Intel Celeron, 512MB RAM
- nodes 4-7: 1.2GHz Intel Pentium 3, 512MB RAM
- Fedora Core 4, kernel 2.6.14
- gcc version 4.0.1

D.1.3 Onyx

Onyx is the main computer science student Linux lab, but it is also a teaching and student research cluster with a dual-processor master node and 32 nodes.

- private gigabit ethernet
- master node: dual 2.4GHz Intel Xeon processors, 3GB RAM
- nodes 01-05: 1.4GHz AMD Duron processors, 512MB RAM
- nodes 06-32: 2.8GHz Intel Pentium 4 w/HT processors, 1GB RAM

Appendix E

CONFIGURATION FILES

Clusmon has three configuration files—one for each daemon. The config files are where settings such as IP addresses, ports, and database information are stored. When a daemon is started, it loads the settings from its respective config file.

E.1 Description and Details

E.1.1 `clusmon.web.config`

All settings in this file are for the database connection. Each of these settings need to be considered when configuring the Clusmon web daemon.

```
#Clusmon WebDaemon config file
database_login=clusmon
database_password=password
database_name=clusmon_data
database_ip=localhost
database_port=3382
```

E.1.2 `clusmon.head.config`

The interval is the most important setting in this file—it determines how often in seconds data is collected. The IP address and ports need to match the settings of the other config files. A good number for the thread_pool is the number of nodes on the cluster. qstat should only be on if pbs is installed and configured. The email settings determine where the warning emails are sent.

```
#Clusmon HeadDaemon config file
interval=10
web_connection_timeout=15
web_ip=127.0.0.1
multicast_group=225.5.5.5
multicast_port=5052
multicast_interface_ip=192.168.0.1
head_to_slave_rmi_port=5055
head_to_web_rmi_port=5051
```

```
thread_pool=60
cpu_temp_threshold=130
qstat=on
history_multiple=6
#Email
smtp=beowulf.boisestate.edu
smtp_port=25
average_temp_email_threshold=100
to=clusteradmin@cs.boisestate.edu
from=noreply@clusmon.com
subject=Beowulf in Danger!!
```

E.1.3 `clusmon.node.config`

The first three settings determine how much data is collected. lm_sensors should only be on if lm_sensors is installed and running. The IP address and ports need to match the settings in the `clusmon.head.config` file. The head daemon connection timeout should be larger then the interval set in the master node. The lm_sensors section of this file has three settings: cpu, other, and fan. These need to be set equal to the labels in the sensors output.

```
#Clusmon Slave config file
lm_sensors=on
memory_data=on
net_data=on
multicast_group=225.5.5.5
multicast_port=5052
head_to_slave_rmi_port=5055
cpu_temp_threshold=140
fan_rpm_threshold=1000
ethernet_interface=eth0
head_connection_timeout=15
#lm_sensors output configuration
cpu:=temp1
other:=temp2
other:=temp3
fan:=fan1
fan:=fan4
```

Appendix F

DATABASE TABLES

F.1 Table Descriptions

`current_stats`

`current_stats` (Figure F.1) is the largest table, and holds most of the individual node data. It includes the node_id, memory, net, jobs, load average, uptime, and temperature data. It is updates at every interval. Node_id is the primary key.

`nodes_info`

`nodes_info` (Figure F.2) holds the IP number, cpu count, and fan information. It holds information that doesn't need to be updates at every interval; but some fields are updated. Node_id is the primary key.

`current_users`

`current_users` (Figure F.3) holds information obtained from the qstat command on the head daemon. It's only populated when the qstat option is turned on. The information is used to display the user name and batch name running on a specific node. Node_id is the primary key.

`past_stats`

`past_stats` (Figure F.4) holds historical data. It's the only table that grows rather than just updates. A new row is inserted every time the update interval reaches a user defined update multiple. The fields are cluster averages, such as average temperatures, usage averages, and even nodes used. The date_time field is a MySQL define field that automatically inserts a timestamp when a new row is added. Though the date_time fields are unique, this table doesn't have an official primary key.

Field	Type	Null	Key	Default	Extra
node_id	varchar(30)		PRI		
total_memory	int(11)			0	
free_memory	int(11)			0	
used_memory	int(11)			0	
net_in_packets	int(11)			0	
net_out_packets	int(11)			0	
net_in_kb	int(11)			0	
net_out_kb	int(11)			0	
jobs_running	int(11)			0	
load_avg	int(11)			0	
uptime	int(11)			0	
temp_cpu1	int(11)			0	
temp_cpu2	int(11)			0	
temp_misc1	int(11)			0	
temp_misc2	int(11)			0	

Figure F.1. `current_stats` database table

Field	Type	Null	Key	Default	Extra
node_id	int(11)		PRI	0	
node_name	varchar(50)				
node_ip	varchar(16)				
cpu_count	int(11)			0	
stopped_fan_mask	int(11)			0	
fan_count	int(11)			0	
total_memory	int(11)			0	
uptime	int(11)			0	
fan1_rpm	int(11)	YES		NULL	
fan2_rpm	int(11)	YES		NULL	
fan3_rpm	int(11)	YES		NULL	
fan4_rpm	int(11)	YES		NULL	
fan5_rpm	int(11)	YES		NULL	
fan6_rpm	int(11)	YES		NULL	

Figure F.2. `nodes_info` database table

Field	Type	Null	Key	Default	Extra
node_id	int(11)	YES		NULL	
node_name	char(25)	YES		NULL	
user_name	char(25)	YES		NULL	
batch_name	char(25)	YES		NULL	

Figure F.3. `current_users` database table

Field	Type	Null	Key	Default	Extra
date_time	timestamp	YES		CURRENT_TIMESTAMP	
avg_temp	int(3)	YES		NULL	
low_temp	int(3)	YES		NULL	
high_temp	int(3)	YES		NULL	
mem_avg	int(2)	YES		NULL	
cpu_avg	int(2)	YES		NULL	
network_avg	int(10)	YES		NULL	
uptime_avg	int(13)	YES		NULL	
nodes_used	int(3)	YES		NULL	

Figure F.4. `past_stats` database table