Autonomic Data Center Thermal Management

Sarah Anne Coe
sarah.anne.coe@gmail.com

Eric Principato
eric@princer.com

Omar Rizwan
omar.rizwan@gmail.com

Katherine Ye
katherine.ye@gmail.com

NJ Governor's School of Engineering & Technology 2011

1 Abstract

Data centers play the critical role of hosting company data. As data centers have increased in size and scope, they have also started to generate increasing amounts of heat, thus requiring increasing amounts of power for self-cooling. Data center cooling now demands increased efficiency and autonomy in order to better protect the environment, improve server stability, and minimize costs. Autonomic computing provides a viable solution: by programming computers to self-manage and self-optimize their processes, they can maximize their own efficiency without human intervention. To explore autonomic approaches to data center environmental management, we simulated a workload on a cluster of eight servers. We developed an autonomic scheduler in Perl to manage these servers and distribute the simulated tasks to minimize heat production. Our scheduler averted dangerous temperature increases on our servers, improving the reliability of these systems.

2 Introduction

To handle the needs of modern networked computing, many organizations around the world have built large-scale data centers with tens of thousands of servers or more [1]. Data centers have evolved over the years to become more efficient, reliable, and cost-effective. These improvements have come from computer professionals' continual search for new ways to improve these centers' functionality.

Environmental control is a key issue associated with data center management: providers must keep their servers cool, or they will overheat and cause service outages. Large data centers pull tens of megawatts of power from the electric grid [2], and they produce heat as a harmful byproduct of normal operation.

If the temperature of air entering the servers exceeds 25 ºC (77 ºF) [3], then the overheated hardware could cause the computers to run more slowly or even break down, which leads to downtime [4]. Downtime is the biggest threat to the operation of modern data centers, because it halts productivity and damages consumer trust. Therefore, data center operators invest money in solutions that reduce heat production and overheating.

Environmental control in data centers typically involves upgrading air conditioners and engineering the airflow of the data center to cool the servers as much as possible. However, enhancing hardware and optimizing external conditions cannot take advantage of resources on the server itself; thus, recent research has investigated software-centric methods of cooling data centers [1]. Autonomic workload management is one such method. By autonomically distributing computations to the coolest available servers, local hot spots are

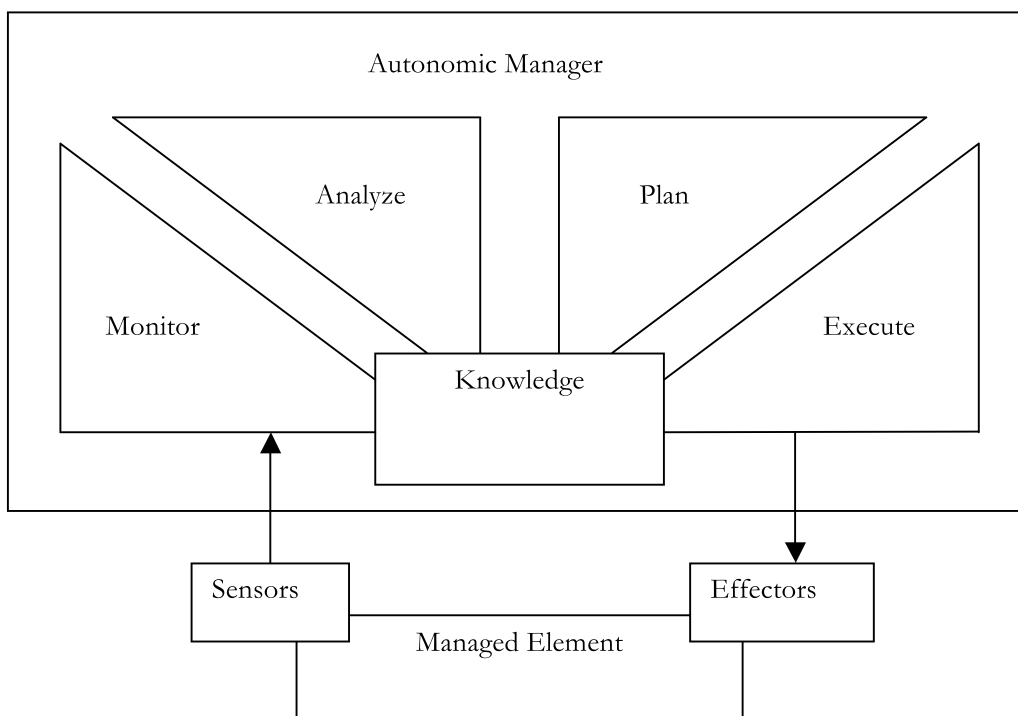


Figure 1: An autonomic element

minimized and the overall ambient temperature decreases.

3 Background

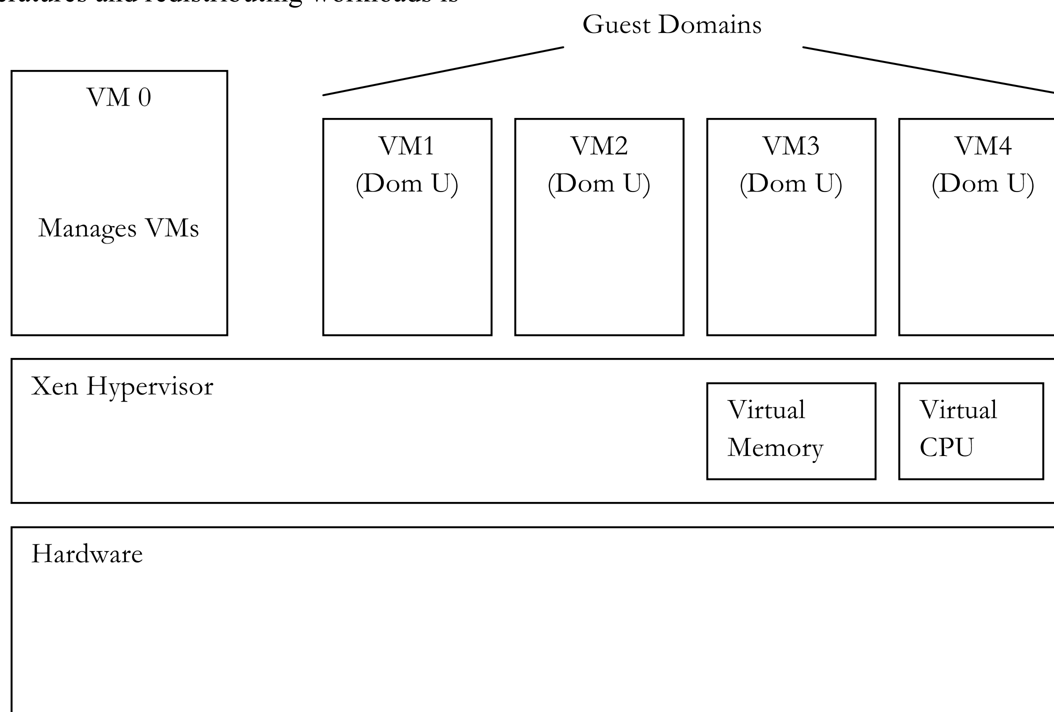
Autonomic computing began in 2001 as an IBM initiative to reduce the increasingly unmanageable complexity of today's computing and data storage systems. It is an approach modeled on the body's autonomic nervous system, which regulates itself according to external conditions by adjusting breathing, blinking, and swallowing rates as well as stimulating the reflexes.

Similarly, autonomic computing systems do not require adjustment or debugging by a systems administrator; they are designed to be "self-configuring, self-healing, self-optimizing, and self-protecting." These properties require autonomous systems to also be capable of "self-awareness, environment-awareness, self-monitoring and self-adjusting." [5] An autonomic "manager" will handle menial tasks delegated to it and also cope with changing conditions, enhancing the efficiency and reliability of data storage and retrieval.

An increase in efficiency greatly benefits data center operators. As processing power has increased, so too has data consumption: the amount of data is doubling roughly twice as fast as the ability to process the data with a fixed number of processors [4]. Yet the typical server runs at less than 50% of capacity due to data center managers' need to preserve extra space for server uptime [4]. Autonomic managing can provide the precision and quickness necessary to juggle the tasks of server workload optimization and thermal management.

Autonomic management is a useful solution to the growing problem of data center thermal management and an important tool for data center administrators. Abstractly, autonomic systems monitor conditions, adapt to changing circumstances, and direct system tasks. To reduce temperatures of a data center, an autonomic strategy of monitoring server

temperatures and redistributing workloads is ideal.



3.1

Figure 2: Xen architecture

Current Work

Current work on data center thermal management focuses on hardware cooling or reducing energy consumption. Rodero et al. [6] propose to manage data center temperatures. Moore et al. [7] similarly propose to accomplish the same task by developing work-scheduling algorithms to facilitate "temperature-aware workload placement." We combine their approaches into a more comprehensive solution to the problem of overheating in data centers. By using Xen to detect dangerously overheated machines and migrating VMs according to server workload, we can better alleviate server workload.

4 Methods

4.1 Platform

This project was executed in a Linux CentOS 5 environment and autonomic components were developed in Perl. Virtual machines to host tasks were set up with Xen 3.0 on eight Rutgers server nodes, as shown in Figure 3 (dual-core CPU, 4 GB RAM, 4 VMs per node). The data were collected and graphed in MATLAB. Each virtual machine's IP address was created in the form 10.0.node.VM (e.g. 10.0.5.3 refers to the 3rd VM on the 5th node).

Xen is a hypervisor inserted as a thin layer of abstraction between a server's hardware and operating system (see Figure 2) [7]. This allows each server to run multiple virtual machines that each act as a physical machine. The hypervisor is a supervisor that shares hardware elements between virtual machines. It allows multiple operating systems to run simultaneously on a server, increasing energy efficiency and decreasing the amount of required hardware [8]. Domain 0, a virtual machine responsible for accessing other virtual machines and accessing the physical hardware, runs on the Xen hypervisor. Other guest domains run on the

server and do not have access to the physical hardware.

We set up and administered the machines remotely using Secure Shell (SSH). SSH allows for remote administration of machines in a cluster [9]. It encrypts data sent between machines, increasing security and preventing unwanted access of clusters from the internet. A public and private key system allowed rapid access and automated control of servers within the system.

4.2 Design

The purpose of our project is to regulate thermal hot-spots in data centers by managing server workloads autonomically. First, we created a monitoring component which collected data on server conditions for analysis. This component periodically checked server temperature readings from internal sensors. It was also able to warn the administrator in the case of critical overheating.

We then developed an autonomic scheduler which would automatically shuffle computational tasks on a cluster of eight servers, with four virtual machines that we set up per server, according to preset guidelines. These guidelines included the demand on a certain server, the server's current operating temperature, and the difficulty of the tasks. The autonomic scheduler analyzed data collected by the autonomic monitor and acted on that data.

The autonomic scheduler's goal was to minimize the overall temperature of each server. It assigned tasks to the server with the lowest current temperature. In addition, the scheduler paused virtual machines running on dangerously overheated servers to reduce their heat production.

The tasks used were chosen from the NAS Parallel Benchmarks set. These benchmarks are a series of programs based on computational fluid dynamics [10] that evaluate computer performance. There are eight benchmark types, each with seven problem size levels.

We implemented these programs in an autonomic scheduler to simulate processes run by users of the network while collecting data on how long each benchmark took to run. These data were used to determine which benchmarks were placed in the scheduler. Most benchmarks used ranged from 10 to 20 minutes with some shorter or longer to model the unpredictability of actual workload conditions. While choosing tasks, it was important to remember that when multiple tasks ran on the same CPU, benchmark completion time could be significantly greater than if the programs ran on independent CPUs.

```
find free VM:
    loop through nodes from 2 to 9
        loop through VMs from 1 to 4
            if this VM on this node is unused:
                run job on this VM on this node
```

Figure 3: Sequential algorithm

```
find free VM:
    do
            node = random node from 2 to 9
            VM = random VM from 1 to 4
        until this VM on this node is not occupied
    run job on this VM on this node
```

Figure 4: Random scheduler

```
find free VM:
    do
        update temperature readings from monitor
        loop through nodes from 2 to 9
            if temperature of this node > 55:
                pause one more VM on this node
            else:
                unpause all VMs on this node

        coolest_node = 2 [by default]
        loop through nodes from 2 to 9
            if temperature of this node < temperature of coolest_node:
                coolest_node = this node

        VM = random VM from 1 to 4
        wait 5 seconds
    while this VM on coolest_node is occupied or temperature of
coolest_node > 55
```

Figure 5: Autonomic algorithm

4.3 Algorithms

First, we tested task scheduling using a naive algorithm which did not account for machine load or monitoring data (Figure 3). Tasks were scheduled node-by-node, with one VM assigned to each submitted task.

We then tested another non-autonomic algorithm, which scheduled tasks to random nodes (Figure 4).

We expected reduced bias toward lower-numbered nodes, but the average temperature increase across all server nodes remained similar.

Finally, we tested task scheduling with an autonomic algorithm which prioritized task scheduling to low-temperature nodes (Figure 5).

This algorithm manages virtual machines and tasks in several different ways. It continually analyzes temperature reports from the monitoring module.

Prior to scheduling tasks, the scheduler ensures the safe operation of all the servers. If any node is operating at a higher temperature than our defined threshold of 55 ºC, the scheduler immediately disables a virtual machine on that node to reduce load. As long as a node's temperature is too high, the scheduler continues pausing virtual machines on each temperature check. Once the temperature has decreased to a safe level, the scheduler unpauses the virtual machines on the node.

After this safety check, we identify the coolest node on which to schedule this task and select a random VM on that node. We always schedule tasks to the coolest node to avoid overheating servers. If the VM chosen on that node is occupied, or if the node is too hot, we wait and try this process again after five seconds.

5 Results

Our first algorithm, the sequential scheduler, biased scheduling toward lower-numbered nodes, since tasks are scheduled to the first

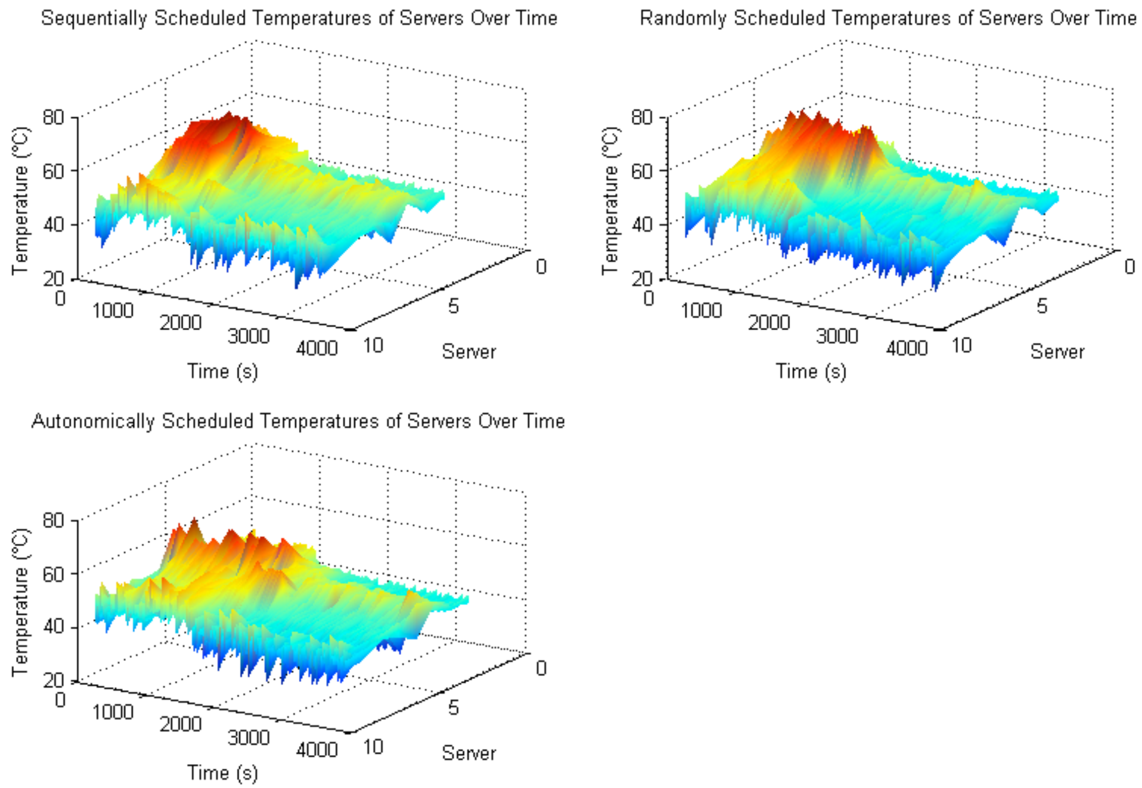


Figure 9: Server temperatures from different scheduling algorithms

available VM on the first available node. A typical task distribution across VMs under this sequential scheduler is shown in Figure 6. Task numbers on each VM and server are listed; cells containing "0" represent unused VMs.

Node	VM1	VM2	VM3	VM4
lagrid02	31	39	44	0
lagrid03	37	13	0	0
lagrid04	21	16	0	0
lagrid05	26	18	0	0
lagrid06	5	32	0	0
lagrid07	40	23	0	0
lagrid08	25	41	0	0
lagrid09	0	43	0	0

Figure 6: Sequential task distribution

Assigning tasks sequentially to VMs, as expected, resulted in significant increases in heat across all nodes, as shown in Figure 9.

The random scheduler created a more equal distribution of tasks across nodes, as in Figure 7.

Node	VM 1	VM 2	VM 3	VM 4
lagrid02	0	0	0	0
lagrid03	0	18	0	16
lagrid04	0	0	23	42
lagrid05	39	34	9	32
lagrid06	21	0	40	37
lagrid07	0	0	13	25
lagrid08	5	0	0	43
lagrid09	31	0	0	0

Figure 7: Sequential task distribution

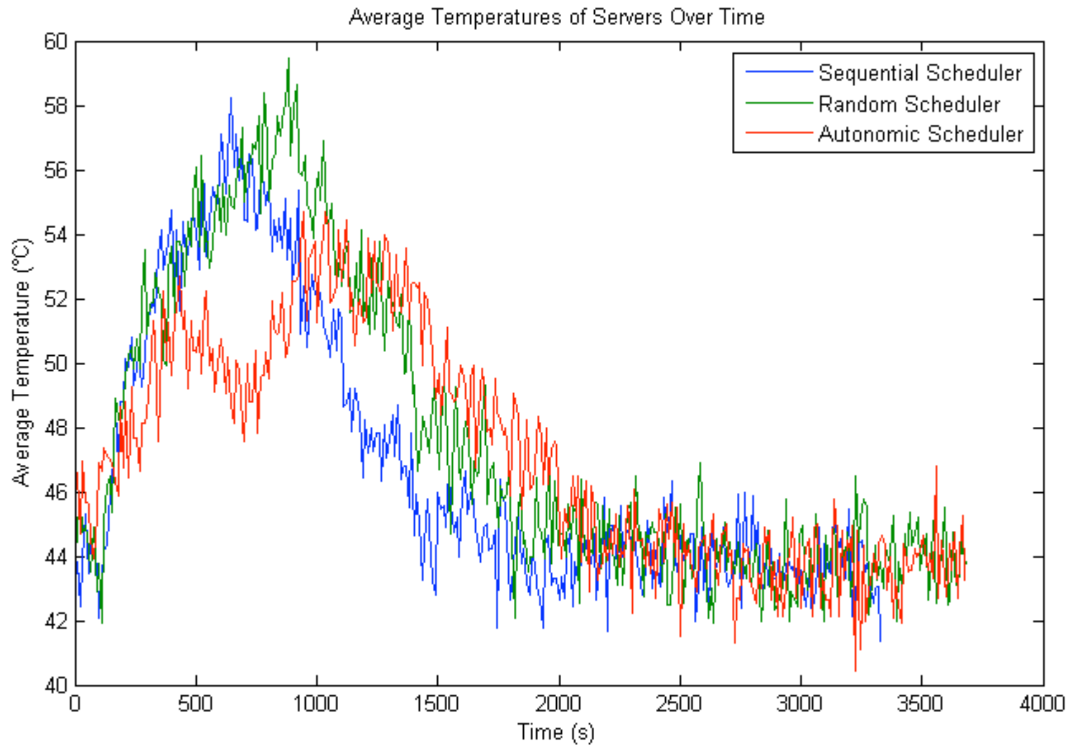


Figure 10: Average temperatures of servers

The more equal distribution of tasks did not, however, lower the average temperature. In fact, random scheduling generally produced higher temperatures on the servers.

Finally, our autonomic scheduler distributed tasks fairly equally among nodes (Figure 8).

Node	VM 1	VM 2	VM 3	VM 4
lagrid02	23	0	37	0
lagrid03	42	0	0	0
lagrid04	0	0	39	0
lagrid05	3	21	41	0
lagrid06	13	0	25	0
lagrid07	0	43	0	0
lagrid08	18	0	31	40
lagrid09	0	26	0	32

Autonomic scheduling reduced the peak temperature of the servers, as indicated by Figure 9. Instead, tasks took longer to complete, although the servers usually ran at lower temperatures.

The autonomic scheduler hollowed out the maximum average temperature, as observed in Figure 10, since it immediately stopped any servers which were running at excessively high temperatures.

6 Analysis/Discussion

Our autonomic software package, consisting of a monitor and scheduler, successfully prevented servers from reaching damaging temperatures, even though it was a relatively simple program. Without the autonomic manager, the servers reached dangerously high temperatures under load; some servers were hotter than 55 ºC for long periods of time.

With the autonomic manager, we observed a noticeable maximum temperature decrease. The program paused virtual machines on servers above the threshold temperature and discouraged overloading already-hot systems. The program ensured that server temperatures rarely exceeded 55 ºC, our preset danger threshold.

Comparing the sequential scheduler algorithm, the random algorithm, and the autonomic algorithm, it is clear from the graphs of temperature vs. time that the autonomic algorithm was able to stabilize the servers and prevent them from experiencing long periods of dangerous overheating. However, it raised the average temperature by about 0.2 ºC (from 46.9 ºC to 46.9 ºC).

7 Conclusion

Autonomic management techniques, a software solution to the problem of data center thermal management, have proven effective in reducing average server temperatures. In addition to being effective, they are also inexpensive: we were able to avoid having to pay for new servers, air conditioning, or shifting servers around to optimize datacenter airflow.

Effective cooling using autonomic managing can significantly improve datacenter productivity. Because 11.5% of servers fail every year at a temperature of 45 ºC [11], if a datacenter has 500 machines, then over 50 machines must be replaced per year. If the temperature is not handled, even more machines will fail. Therefore, an autonomic system can save money through both energy savings and maintenance savings.

8 Acknowledgements

We would like to thank Dr. Ivan Rodero and Prof. Parashar, our mentors, and Cristina Sorice, our RTA, for teaching us about autonomic computing and helping to edit our paper. We also acknowledge our debt to those who made our project possible: Dean Ilene Rosen, Program Coordinator Jean Patrick Antoine, and Head Counselor Daniel Cobar. We thank our 2011 sponsors for their support of GSET: Rutgers University, the Rutgers University School of Engineering, Morgan Stanley, the State of New Jersey, Lockheed Martin, Automated Control Concepts Inc., Silver Line Building Products, Sharon Ma and Nan Yao (parents of Johnathan Yao, GSET 2010), the Tomasetta family, and Laura Overdeck. Finally, we express our gratitude to the NJ Governor's School of Engineering and Technology and its Board of Overseers.

9 References

[1] Moore, Justin, Jeff Chase, Parthasarathy Ranganathan, and Ratnesh Sharma. "Making Scheduling "Cool": Temperature-Aware Workload Placement in Data Centers." *USENIX Annual Technical Conference*. 2005. Web. 13 July 2011.

[2] Tung, Teresa. "Data Center Energy Forecast." *Silicon Valley Leadership Group*. 2008. Web. 13 July 2011.

[3] ASHRAE Publication, "ASHRAE Environmental Guidelines for Datacom Equipment", Atlanta. 2008. Web. 13 July 2011.

[4] "The Problem of Thermal Management." *Sharkrack*. 2008. Web. 13 July 2011.

[5] "Autonomic Computing | Overview | The Solution." *IBM Research*. IBM, 23 Feb. 2001. Web. 13 July 2011. <http://www.research.ibm.com/autonomic/overview/solution.html>.

[6] Rodero, Ivan, Eun Kyung Lee, Dario Pompili, Manish Parashar, Marc Gamell, and Renato J. Figueiredo. "Towards Energy-Efficient Reactive Thermal Management in Instrumented Datacenters," Proc. of IEEE/ACM International Conference on Energy Efficient Grids, Clouds and Clusters

Workshop (E2GC2), Brussels, Belguim, October 2010. Print.

[7] "Xen Hypervisor." *Welcome to Xen.org, Home of the Xen® Hypervisor, the Powerful Open Source Industry Standard for Virtualization.* Xen, 2011. Web. 13 July 2011. <http://www.xen.org/products/xenhyp.html>.

[8] "Virtual Machines, Virtual Server, Virtual Infrastructure." *VMware Virtualization Software for Desktops, Servers & Virtual Machines for Public and Private Cloud Solutions*. Web. 16 July 2011. <http://www.vmware.com/virtualization/virtual-machine.html>.

[9] "How to Use SSH Secure Shell: What Is SSH?" *University Information Services (UIS)*. Georgetown University. Web. 13 July 2011. <http://uis.georgetown.edu/software/documentation/ssh/>.

[10] "NAS Parallel Benchmarks Changes." *NASA Advanced Supercomputing (NAS) Division Home Page*. NASA, June-July 2010. Web. 13 July 2011. <http://www.nas.nasa.gov/Resources/Software/npb.html>.

[11] Maier, Gerald. "Datacenter Room Temperature of 45 Degrees Celsius – Cover Higher Server Failure Rates by Significant Savings from Cooling." *Datacenter & IT-Operations Management* (2011): n. pag. Web. 21 Jul 2011. <http://dcito.wordpress.com/2011/03/04/datacenter-room-temperature-of-45-degrees-celsius-cover-higher-server-failure-rates-by-significant-savings-from-cooling/>.

10 Appendix

10.1 Temperature monitor

```
#!/usr/bin/perl
# use strict;
use warnings;

$SIG{CHLD} = 'IGNORE';
open(TEMPOUT, ">temperatures.out");
my @tempsHistory = ();
while (1) {
	my @temps = ();
	for (my $count = 2; $count <=
9; $count++) {
		pipe *{$count},RETHAND;
		unless (fork()) {
			open (SSHOUT,
"ssh lagrid0$count sensors |");
			my $temp;

			while (<SSHOUT>)
{
				if ($_ =~
m/CPU Temp/) {
					$temp
= $_;

					$temp
=~ s/CPU Temp:  //g;
					$temp
=~ s/\+//g;
					$temp
=~ s/°C.*$//g;
				}
			}

			print RETHAND
$temp;
			exit();
		}
	}

	print TEMPOUT time;
	print time;
	for (my $count = 2; $count <=
9; $count++) {
		my $temp = <$count>;
		chomp($temp);

		print TEMPOUT ",$temp";
		print ",$temp";
		if ($temp > 45 and
$tempsHistory[-1] and
$tempsHistory[-2] and
```

```
                  $tempsHistory[-
1][$count] > 45 and
                  $tempsHistory[-
2][$count] > 45) {
                  # print "ALARM on
lagrid0$count \n";
            }
            $temps[$count] = $temp;
      }

      push(@tempsHistory, [ @temps
]);
      while (scalar @tempsHistory >
4) {
            shift @tempsHistory;
      }

      print TEMPOUT "\n";
      print "\n";
      sleep 10;
}
```

10.2 Scheduler

```
#!/usr/bin/perl
use strict;
use warnings;

use POSIX qw(ceil);
use Parallel::ForkManager;

use IPC::Open3;
use Symbol qw(gensym);
use IO::File;

use Data::Dumper;

use constant NOT_STARTED => -1;
use constant RUNNING => 1;
use constant FAILED => -2;
use constant DONE => 2;

my @node_vm_jobs = ();
my @jobs = ();

my @queued_jobs = ();

my @node_temps = ();
my @node_pauses = ();
my @vms;
for (my $vm = 1; $vm <= 4; $vm++) {
      $vms[$vm] = 0;
}

for (my $node = 2; $node <= 9;
$node++) {
      $node_vm_jobs[$node] = [ @vms
];
      $node_pauses[$node] = 0;
}

my $pm = new
Parallel::ForkManager(32);

$pm->run_on_finish(
      sub {
            my ($pid, $exit_code,
$job_id, $exit_signal, $core_dump,
$data) = @_;

            my $job =
$jobs[$job_id];
            my $node = $job-
>{node};
            my $vm = $job->{vm};

            if ($exit_code) {
                  $job->{status} =
FAILED;

      failed_job($job_id);

                  print time.":
Process done: Job $job_id on
10.0.$node.$vm failed!\n";

            } else {
                  $job->{status} =
DONE;
                  $job->{time} =
$data;
                  print time.":
Process done: Job $job_id on
10.0.$node.$vm succeeded!\n";
            }

      $node_vm_jobs[$node][$vm] =
0;
```

```
      }
      );

open(JOBLIST, "<jobs.txt");
while (<JOBLIST>) {
      chomp($_);
      add_job($., $_);
}

my $when_no_queue = 0;
while (1) {
      print "Beginning round, task
queue size: ".@queued_jobs."\n";

      if (scalar @queued_jobs) {
            $when_no_queue = 0;

            my $job =
pop(@queued_jobs);
            next_job($job);
      } else {
            print time.": Queue
empty!\n";

            my $still_running = 0;
            for (my $node = 2;
$node <= 9; $node++) {
                  for (my $vm = 1;
$vm <= 4; $vm++) {
                        if
($node_vm_jobs[$node][$vm] > 0) {

      $still_running = 1;
                        }
                  }
            }

            unless ($still_running)
{
                  last;
            }
      }

      sleep 5;
}

print time.": Queue loop
completed.\n";
$pm->wait_all_children;
print time.": All processes
completed.\n";

sub update_temperatures {
      my $temp_reading = `tail -n1
temperatures.out`;
      chomp($temp_reading);

      @node_temps = split(/,/,
$temp_reading);
      unshift(@node_temps, -1); #
unshift so that array indices are
2-9
}

sub manage_node_temperatures {
      print "Node temperatures: ";
      for (my $node = 2; $node <=
9; $node++) {
            print
"$node_temps[$node]";
            if ($node_temps[$node]
> 50) {

      $node_pauses[$node]++;
                  print ": ALARM,
PAUSING VM ".$node_pauses[$node];

                  if
($node_pauses[$node] == 1) {
                        system("ssh
10.0.0.$node \"sudo xm pause
Fedora12\"");
                  } elsif
($node_pauses[$node] == 2) {
                        system("ssh
10.0.0.$node \"sudo xm pause
Fedora12-1\"");
                  } elsif
($node_pauses[$node] == 3) {
                        system("ssh
10.0.0.$node \"sudo xm pause
Fedora12-2\"");
                  } elsif
($node_pauses[$node] == 4) {
                        system("ssh
10.0.0.$node \"sudo xm pause
Fedora12-3\"");
                  }
            } else {
```

```
                  if
($node_pauses[$node] > 0) {

      $node_pauses[$node] = 0;

                        print ":
SAFE, UNPAUSING";
                        system("ssh
10.0.0.$node \"sudo xm pause
Fedora12 && sudo xm unpause
Fedora12-1 && sudo xm unpause
Fedora12-2 && sudo xm unpause
Fedora12-3\"");
                  }
            }
            print ", ";
      }
      print "\n";
}

sub add_job {
      my ($job_id, $job_task) = @_;

      my $job = {
            id => $job_id,

            task => $job_task,

            status => NOT_STARTED,

            pid => 0,

            node => 0,
            vm => 0,

            time => 0
      };

      $jobs[$job_id] = $job;
      unshift(@queued_jobs, $job);
}

sub failed_job {
      my ($job_id) = @_;

      push(@queued_jobs,
$jobs[$job_id]);
}

sub next_job {
      my ($job) = @_;

      my ($node, $vm);
      do {
            ($node, $vm) =
find_free_vm();
      } while (!run_job($node, $vm,
$job->{id}, $job->{task}));

      print_status();
      print time.": Round
complete!\n\n";
}

sub find_free_vm {
      my ($node, $vm);

      if ($ARGV[0] eq "-a") {
            # AUTONOMIC ALGORITHM
            # do {
            #     $node =
int(rand(8)+2);
            #     $vm =
int(rand(4)+1);
            #     if
($node_temps[$node] > 50) {
            #           print
"Broken threshold for VM $vm on
node $node\n";
            #     }
            # } while
(($node_temps[$node] > 50) ||
($node_vm_jobs[$node][$vm]));

            # print "10.0.$node.$vm
is free\n";
            # return ($node, $vm);

            my $coolest_node;
            do {

      update_temperatures();

      manage_node_temperatures();

                  $coolest_node =
2;
                  for ($node = 2;
$node <= 9; $node++) {
```

```
                        if
($node_temps[$node] <
$node_temps[$coolest_node]) {

      $coolest_node = $node;
                        }
                  }
                  print time.":
Coolest node is $coolest_node\n";

                  $vm =
int(rand(4)+1);
                  sleep 5;
            } while
(($node_vm_jobs[$coolest_node][$vm]
) || ($node_temps[$coolest_node] >
50));

            print
"10.0.$coolest_node.$vm is free\n";
            return ($coolest_node,
$vm);
      } elsif ($ARGV[0] eq "-R") {
            # RANDOM ALGORITHM
            do {
                  $node =
int(rand(8)+2);
                  $vm =
int(rand(4)+1);
                  sleep
            } while
($node_vm_jobs[$node][$vm]);

            print "10.0.$node.$vm
is free\n";
            return ($node, $vm);
      } elsif ($ARGV[0] eq "-s") {
            # SEQUENTIAL ALGORITHM
            for ($vm = 1; $vm <= 4;
$vm++) {
                  for ($node = 2;
$node <= 9; $node++) {
                        unless
($node_vm_jobs[$node][$vm]) {
                              print
"10.0.$node.$vm is free\n";

      return ($node, $vm);
                        }
                  }
            }
      } else {
            print "Usage:
scheduler.pl -a|-R|-s\n";
            exit(1);
      }
      print "ERROR, no VMs free.";
}

sub run_job {
      my($node, $vm, $job_id,
$job_task) = @_;

      if
($node_vm_jobs[$node][$vm]) {
            print "$vm is occupied,
uh-oh\n";
            return 0;
      }

      my $ip = "10.0.$node.$vm";
      my $command = "ssh $ip
~/benchmarks/$job_task\n";

      print time.": Running job
$job_id on $ip\n";
      print "Command: $command\n";

      my $job = $jobs[$job_id];
      $job->{status} = RUNNING;
      $job->{node} = $node;
      $job->{vm} = $vm;

      $node_vm_jobs[$node][$vm] =
$job_id;
      $job->{pid} = $pm-
>start($job_id) and return 1;
      # child process runs from now
on and manages the job

      local *CATCHERR = IO::File-
>new_tmpfile;
      my $ssh_pid = open3(gensym,
\*CATCHOUT, ">&CATCHERR",
"$command");

      while (<CATCHOUT>) {
            if ($_ =~ m/Time in
seconds/) {
                  my $time = $_;
```

```
			$time =~ s/\sTime 
in seconds =\s//g;
			print time.": Job 
$job_id on $ip SUCCEEDED--time: 
$time\n";
			$pm->finish(0, 
$time);
		}
	}

	waitpid($ssh_pid, 0);

	seek CATCHERR, 0, 0;
	while (<CATCHERR>) {
		print $_;
		if ($_ =~ m/Connection 
timed out/) {
			print time.": Job 
$job_id on $ip FAILED: connection 
timed out.\n";
			$pm->finish(1);
		}

		elsif ($_ =~ m/No route 
to host/) {
			print time.": Job 
$job_id on $ip FAILED: no route to 
host.\n";
			$pm->finish(2);
		}

		else {
			print time.": Job 
$job_id on $ip FAILED: unknown 
error.\n";
			$pm->finish(3);
		}
	}

	$pm->finish(4);
}

sub print_status {
	print "node_vm_jobs 
status:\n";
	print "NODE\tVM 1\tVM 2\tVM 
3\tVM 4\n";
	for (my $node = 2; $node <= 
9; $node++) {
		print "$node:\t";
		for (my $vm = 1; $vm <= 
4; $vm++) {
			print 
"$node_vm_jobs[$node][$vm]\t";
		}
		print "\n";
	}
	print "\n";

	# print "jobs status:\n";
	# foreach (@jobs) {
	#	if (defined $_) {
	#		my %job = %{ $_ 
};
	#		while ( my ($k, 
$v) = each %job ) {
	#			print "$k 
=> $v\n";
	#		}
	#		print "\n";
	#	}
	# }
}
```