Mercury and Freon: Temperature Emulation and Management for Server Systems *

Taliver Heath
Dept. of Computer Science
Rutgers University
taliver@cs.rutgers.edu

Ana Paula Centeno
Dept. of Computer Science
Rutgers University
anapaula@cs.rutgers.edu

Pradeep George
Dept. of Mechanical Engineering
Rutgers University
pradeepg@rci.rutgers.edu

Luiz Ramos
Dept. of Computer Science
Rutgers University
luramos@cs.rutgers.edu

Yogesh Jaluria
Dept. of Mechanical Engineering
Rutgers University
jaluria@jove.rutgers.edu

Ricardo Bianchini
Dept. of Computer Science
Rutgers University
ricardob@cs.rutgers.edu

Abstract

Power densities have been increasing rapidly at all levels of server systems. To counter the high temperatures resulting from these densities, systems researchers have recently started work on *software-based thermal management*. Unfortunately, research in this new area has been hindered by the limitations imposed by simulators and real measurements. In this paper, we introduce Mercury, a software suite that avoids these limitations by accurately emulating temperatures based on simple layout, hardware, and component-utilization data. Most importantly, Mercury runs the entire software stack natively, enables repeatable experiments, and allows the study of thermal emergencies without harming hardware reliability. We validate Mercury using real measurements and a widely used commercial simulator. We use Mercury to develop Freon, a system that manages thermal emergencies in a server cluster without unnecessary performance degradation. Mercury will soon become available from http://www.darklab.rutgers.edu.

Categories and Subject Descriptors D.4 [*Operating systems*]: Miscellaneous

General Terms Design, experimentation

Keywords Temperature modeling, thermal management, energy conservation, server clusters

1. Introduction

Power densities have been increasing rapidly at all levels of server systems, from individual devices to server enclosures to machine rooms. For example, modern microprocessors, high-performance disk drives, blade server enclosures, and highly populated computer racks exhibit power densities that have never been seen before. These increasing densities are due to increasingly power-hungry hardware components, decreasing form factors, and tighter packing. High power densities entail high temperatures that now must be countered by substantial cooling infrastructures. In fact, when hundreds, sometimes thousands, of these components are racked close together in machine rooms, appropriate cooling becomes the main concern.

The reason for this concern is that high temperatures decrease the reliability of the affected components to the point that they start to behave unpredictably or fail altogether. Even when components do not misbehave, operation outside the range of acceptable temperatures causes mean times between failures (MTBFs) to decrease exponentially [1, 7]. Several factors may cause high temperatures: hot spots at the top sections of computer racks, poor design of the cooling infrastructure or air distribution system, failed fans or air conditioners, accidental overload due to hardware upgrades, or degraded operation during brownouts. We refer to these problems as "thermal emergencies". Some of these emergencies may go undetected for a long time, generating corresponding losses in reliability and, when components eventually fail, performance.

Recognizing this state of affairs, systems researchers have recently started work on *software-based thermal management*. Specifically, researchers from Duke University and Hewlett-Packard have examined temperature-aware workload placement policies for data centers, using modeling and a commercial simulator [21]. Another effort has begun investigating temperature-aware disk-scheduling policies, using thermal models and detailed simulations of disk drives [12, 16]. Rohou and Smith have implemented and experimentally evaluated the throttling of activity-intensive tasks to control processor temperature [26]. Taking a different approach, Weissel and Bellosa have studied the throttling of energy-intensive tasks to control processor temperature in multi-tier services [32].

Despite these early initiatives, the infrastructure for software-based thermal management research severely hampers new efforts. In particular, both real temperature experiments and temperature simulators have several deficiencies. Working with real systems requires heavy instrumentation with (internal or external) sensors collecting temperature information for every hardware component and air space of interest. Hardware sensors with low resolution and poor precision make matters worse. Furthermore, the environment

* This research has been supported by NSF under grant #CCR-0238182 (CAREER award).

ASPLOS'06 October 21–25, 2006, San Jose, California, USA.

where the experiments take place needs to be isolated from unrelated computations or even trivial "external thermal disruptions", such as somebody opening the door and walking into the machine room. Under these conditions, it is very difficult to produce repeatable experiments. Worst of all, real experiments are inappropriate for studying thermal emergencies. The reason is that repeatedly inducing emergencies to exercise some piece of thermal management code may significantly decrease the reliability of the hardware.

In contrast, temperature simulators do not require instrumentation or environment isolation. Further, several production-quality simulators are available commercially, e.g. Fluent [9]. Unfortunately, these simulators are typically expensive and may take several hours to days to simulate a realistic system. Worst of all, these simulators are not capable of executing applications or any type of systems software; they typically compute steady-state temperatures based on a fixed power consumption for each hardware component. Other simulators, such as HotSpot [30], do execute applications (bypassing the systems software) but only model the processor, rather than the entire system.

To counter these problems, we introduce *Mercury*, a software suite that emulates system and component temperatures in single-node or clustered systems in a repeatable fashion. As inputs, Mercury takes simple heat flow, air flow, and hardware information, as well as dynamic component utilizations. To enable thermal management research, the emulation is performed at a coarse grain and seeks to approximate real temperatures to within a few degrees while providing trend-accurate thermal behavior. Although this is sometimes unnecessary, users can improve accuracy by calibrating the inputs with a few real measurements or short simulations.

Mercury provides the same interface to user-level software as real thermal sensors, so it can be linked with systems software to provide temperature information in real time. In fact, in our approach *the entire software stack runs natively (without noticeable performance degradation) and can make calls to Mercury on-line*, as if it were probing real hardware sensors. Our suite also provides mechanisms for explicitly changing temperatures and other emulation parameters at run time, allowing the simulation of thermal emergencies. For example, one can increase the inlet air temperature midway through a run to mimic the failure of an air conditioner. Finally, Mercury is capable of computing temperatures from component-utilization traces, which allows for fine-tuning of parameters without actually running the system software. In fact, replicating these traces allows Mercury to emulate large cluster installations, even when the user's real system is much smaller.

We validate Mercury using measurements and simulations of a real server. The comparison against Fluent, a widely used commercial simulator, demonstrates that we can approximate steady-state temperatures to within 0.5^oC, after calibrating the inputs provided to Mercury. The comparison against the real measurements shows that dynamic temperature variations are closely emulated by our system as well. We have found emulated temperatures within 1^oC of the running system, again after a short calibration phase.

We are using Mercury to investigate policies for managing thermal emergencies without excessive (throughput) performance degradation. In particular, we are developing *Freon*, a system that manages component temperatures in a server cluster fronted by a load balancer. The main goal of Freon is to manage thermal emergencies without using the traditional approach of turning off the affected servers. Turning servers off may degrade throughput unnecessarily during high-load periods. Given the direct relationship between energy consumption and temperature, as an extension of Freon, we also develop a policy that combines energy conservation and thermal management. Interestingly, the combined policy does turn servers off when this is predicted not to degrade throughput.

To accomplish its goals, Freon constantly monitors temperatures and dynamically adjusts the request distribution policy used by the load balancer in response to thermal emergencies. By manipulating the load balancer decisions, Freon directs less load to hot servers than to other servers. Using our emulation suite, we have been able to completely develop, debug, and evaluate Freon.

In summary, this paper makes two main contributions:

- We propose Mercury, a temperature emulation suite. Mercury simplifies the physical world, trading off a little accuracy (at most 1^oC in our experiments) for native software stack execution, experiment repeatability, and the ability to study thermal emergencies; and
- We propose Freon, a system for managing thermal emergencies without unnecessary performance degradation in a server cluster. In the context of Freon, we propose the first policy to combine energy and thermal management in server clusters.

The remainder of the paper is organized as follows. The next section describes the details of the Mercury suite. Section 3 presents our validation experiments. Section 4 describes the Freon policies. Section 5 demonstrates the use of Mercury in the Freon evaluation. Section 6 overviews the related work. Finally, Section 7 discusses our conclusions and the limitations of our current implementations.

2. The Mercury Suite

In this section we describe the physical model that Mercury emulates, overview the Mercury inputs and emulation approach, and detail our current implementation.

2.1 Physics

Modeling heat transfer and air flow accurately is a complex proposition. To create a very accurate simulation from basic properties, Mechanical Engineering tools must simulate everything from wall roughness to fluctuations of the air density to the detailed properties of the air flow. We believe that this level of detail is unnecessary for most software-based thermal management research, so we have simplified our model of the physical world to a few basic equations. This is the key insight behind Mercury.

Basic terminology. Since many systems researchers are unfamiliar with engineering and physics terminology, we will first define some basic concepts. The *temperature* of an object is a measure of the internal energy present in that object. Temperature is typically measured in degrees Celsius or Kelvin. *Heat* is energy that is transferred between two objects or between an object and its environment. Heat is measured in the same units as energy, such as Joules or Wh.

Conservation of energy. One of the most basic equations in heat transfer is the conservation of energy, which translates into the conservation of heat in our case. There are two sources of heat for an object in our system: it converts power into heat while it performs work, and it may gain (lose) heat from (to) its environment. This can be expressed as:

$$Q_{gained} = Q_{transfer} + Q_{component} \quad (1)$$

where $Q_{transfer}$ is the amount of heat transferred from/to the component during a period of time and $Q_{component}$ is the amount of heat produced by performing work during a period of time. Next we define these two quantities in turn.

Newton's law of cooling. Heat is transferred in direct proportion to the temperature difference between objects or between an object and its environment. More formally:

$$Q_{transfer,1\rightarrow 2} = k \times (T_1 - T_2) \times time \quad (2)$$

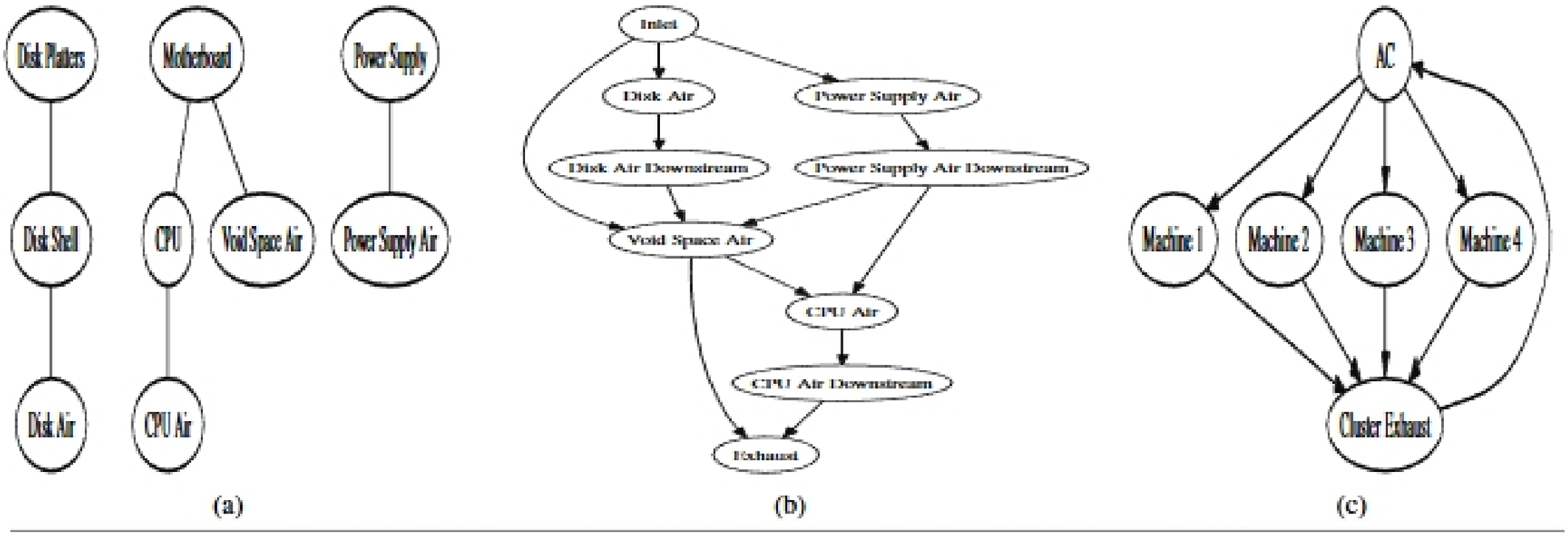

Figure 1. Example intra-machine heat-flow (a), intra-machine air-flow (b), and inter-machine air-flow (c) graphs.

where $Q_{transfer,1\to2}$ is the amount of heat transferred between objects 1 and 2 or between object 1 and its environment during a period of time, T_1 and T_2 are their current temperatures, and k is a constant that embodies the heat transfer coefficient and the surface area of the object. The k constant can be approximated by experiment, simulation, or rough numerical approximation using typical engineering equations. However, in reality, k can vary with temperature and air-flow rates. We assume that k is constant in our modeling because the extra accuracy of a variable k would not justify the complexity involved in instantiating it for all temperatures and rates.

Energy equivalent. The heat produced by a component essentially corresponds to the energy it consumes, so we define it as:

$$Q_{component} = P(utilization) \times time \quad (3)$$

where $P(utilization)$ represents the average power consumed by the component as a function of its utilization.

For the components that we have studied in detail so far, a simple linear formulation has correctly approximated the real power consumption:

$$P(utilization) = P_{base} + utilization \times (P_{max} - P_{base}) \quad (4)$$

where P_{base} is the power consumption when the component is idle and P_{max} is the consumption when the component is fully utilized.

Researchers have used similar high-level formulations to model modern processors and server-style DRAM subsystems [8]. However, our default formulation can be easily replaced by a more sophisticated one for components that do not exhibit a linear relationship between high-level utilization and power consumption. For example, we have an alternate formulation for Q_{CPU} that is based on hardware performance counters. (Although our descriptions and results assume the default formulation, we do discuss this alternate formulation in Section 2.3.)

Heat capacity. Finally, since pressures and volumes in our system are assumed constant, the temperature of an object is directly proportional to its internal energy. More formally, we define the object's temperature variation as:

$$\Delta T = \frac{1}{m\,c} \times \Delta Q \quad (5)$$

where m is the mass of the object and c is its specific heat capacity.

2.2 Inputs and Temperature Emulation

Mercury takes three groups of inputs: heat- and air-flow graphs describing the layout of the hardware and the air circulation, constants describing the physical properties and the power consumption of the hardware components, and dynamic component utilizations. Next, we describe these groups and how Mercury uses them.

Graphs. At its heart, Mercury is a coarse-grained finite element analyzer. The elements are specified as vertices on a graph, and the edges represent either air-flow or heat-flow properties. More specifically, Mercury uses three input graphs: an inter-component heat-flow graph, an intra-machine air-flow graph, and possibly an inter-machine air-flow graph for clustered systems. The heat-flow graphs are undirected, since the direction of heat flow is solely dependent upon the difference in temperature between each pair of components. The air-flow graphs are directed, since fans physically move air from one point to another in the system.

Figure 1 presents one example of each type of graph. Figure 1(a) shows a heat-flow graph for the real server we use to validate Mercury in this paper. The figure includes vertices for the CPU, the disk, the power supply, and the motherboard, as well as the air around each of these components. As suggested by this figure, Mercury computes a single temperature for an entire hardware component, which again is sufficient for whole-system thermal management research. Figure 1(b) shows the intra-machine air-flow for the same server. In this case, vertices represent air regions, such as inlet air and the air that flows over the CPU. Finally, Figure 1(c) shows an inter-machine air flow graph for a small cluster of four machines. The vertices here represent inlet and outlet air regions for each machine, the cold air coming from the air conditioner, and the hot air to be chilled. The graph represents the ideal situation in which there is no air recirculation across the machines. Recirculation and rack layout effects can also be represented using more complex graphs.

Constants. The discussion above has not detailed how the input graphs' edges are labeled. To label the edges of the heat-flow graph and instantiate the equations described in the previous subsection, Mercury takes as input the heat transfer coefficients, the components' specific heat capacities, and the components' surface areas and masses. The air-flow edges are labeled with the fraction of air that flows from one vertex to another. For example, the air-flow edge from the "Disk Air Downstream" vertex to the "CPU Air" vertex in Figure 1(b) specifies how much of the air that flows over the disk flows over the CPU as well. (Because some modern fans change speeds dynamically, we need a way to adjust the air-flow fractions accordingly. The thermal emergency tool described in Section 2.3 can be used for this purpose.) Besides these con-