Automating CPU Dynamic Thermal Control for High Performance Computing

Ghazanfar Ali*, Lowell Wofford†, Christopher Turner*, and Yong Chen*

* Texas Tech University, Lubbock, TX
Email: {ghazanfar.ali, christopher.turner, yong.chen}@ttu.edu

† Ultrascale Systems Research Center, Los Alamos National Laboratory, Los Alamos, NM
Email:lowell@lanl.gov

***Abstract*—In a production high-performance computing (HPC) data center, numerous factors, including workload compute intensity, cooling infrastructure failure, and the use of economized cooling can substantially increase the CPU temperature. CPU thermal design-related studies have shown that slight variances in the operational temperature can significantly impact the lifetime, durability, and performance of a CPU. Therefore, it is critical to monitor and control the operating temperature of the CPU. In this study, we design an automated and continuous CPU thermal monitoring and control methodology to maintain and control a healthy CPU thermal state. This research utilizes the Redfish protocol to monitor the CPU temperature and dynamic voltage frequency scaling to control the temperature. We developed a reference implementation and evaluated our methodology using a cluster of 150 Raspberry Pi3 nodes. We performed extensive CPU thermal analyses in different scenarios. We analyzed how quickly a CPU can attain the maximum temperature under 100% load at room temperature. Based on our experiments, the temperature of a CPU with 100% load can increase to ∼72°C (161.6°F) and ∼86°C (186.8°F) with the lowest and highest CPU frequency configurations, respectively. We analyzed the impact of applying thermal control at eight temperature configurations on the thermal and frequency scaling behavior of a CPU. We observed that applying thermal control at lower temperature configurations (e.g., 70°C (158°F)) is a better configuration for healing an overheated CPU. As a result of the proposed model, the CPU operating at normal temperature will consume comparatively less energy, deliver higher performance, and augment its durability.**

***Index Terms*—CPU Temperature, Automation, HPC, Data Center, Kraken, Dynamic Voltage and Frequency Scaling, Powersave, Performance, Dynamic Thermal Control, Redfish, DVFS, Kraken, Computing Cluster Dynamic Thermal Control, Dynamic Voltage and Frequency Scaling, Data Center Automation, High Performance Computing**

I. Introduction

Thermal dissipation is one of the perennial issues in operating a high performance computing (HPC) data center. Since the inception of large-scale computing systems, thermal control has been extensively studied from different perspectives, including mechanical and software-based mechanisms [1]. Mechanical cooling solutions (e.g., air cooling, liquid cooling) have effectively addressed the room- and rack-level thermal control. However, these solutions are not sufficient to control component-level temperature—particularly for CPUs.

As an after-effect of post-Moore's Law and Pollack's rule [2], the current trend in microprocessor architecture is increasingly based on multi-core and many-core paradigms to deliver performance near exa-scale [3] [4]. Clearly, more cores will draw more power, dramatically increasing the average heat flux, i.e., the power dissipated per unit die area on the CPU chip. Higher heat flux can increase the operating temperature of the CPU significantly [4]. Existing studies, for example [5], identified that CPU temperature management is an increasingly daunting issue in large-scale HPC systems due to inordinate power consumption.

In addition to the CPU thermal issues at the design level, pertinent production level factors can directly or indirectly increase the CPU's operating temperature. These factors include transient changes in workloads, involvement of economizers [6], malfunctioning of the CPU fan, and heating, ventilation, air-conditioning, and cooling (HVAC) failure. Studies revealed that the CPU's operating temperature has a significant impact in a myriad of ways. First, the CPU's operating temperature can increase or decrease its lifespan exponentially depending upon its operational temperature. For example, with a difference of 10-15°C above or below CPU normal operating temperature, a CPU can approximately halve or double its lifespan, respectively [4]. Second, running the CPU at lower operating temperature ranges enables the CPU to deliver better performance due to minimal power leakage [4]. Third, the CPU running at higher temperature ranges can cause additional power consumption [7]. The relationship between power consumption and CPU temperature can be derived from the study [4] as follows:

$$Power \cdot \theta_{ja} = T_j - T_a, \tag{1}$$

where T_j is the die temperature (in this study, CPU temperature is synonymous with CPU die temperature), T_a is the ambient temperature, and $Power$ is the power consumed by the CPU. By keeping T_a and θ_{ja} constant, $Power$ is directly proportional to T_j. In other words, CPU on-chip temperature can be controlled by reducing the power consumption.

Furthermore, the Arrhenius equation has been widely studied and applied to predict the influence of temperature on chemical and biological processes [8]. The Arrhenius-based equation has also been redesigned to estimate the impact of steady-state temperature on the failure rate of electronic devices [9], i.e.,

$$\lambda = \lambda_{ref} \cdot \exp\left(-\frac{E_{dev}}{k \cdot T}\right), \tag{2}$$

where λ, λ_{ref}, E_{dev}, k, and T represent failure rate, reference failure rate, device activation energy, Boltzmann's constant, and steady-state absolute temperature, respectively. The inverse of λ is called mean-time-to-failure (MTTF) [9], i.e.

$$MTTF = \frac{1}{\lambda}. \tag{3}$$

Equations 2 and 3 show that the device's MTTF is inversely proportional to the device operating temperature.

To maintain a healthy CPU temperature and heal an overheated CPU, in this research, we design and implement a methodology to automate a CPU's thermal control in HPC systems. This methodology is implemented in a framework called Kraken for automation and control functionalities and leverages the CPU's frequency scaling mechanisms to control CPU thermal conditions. Overall, this study presents two techniques in the automation of healing of the overheated CPU. First, it monitors the CPU thermal state and the operating CPU frequency simultaneously. Second, it mutates the CPU frequency to heal the CPU temperature when it is overheated. We have tested this methodology on real-world CPUs.

The core contributions of this work include: 1) design of a methodology for monitoring and control of CPU thermal and frequency states and its implementation in the Kraken framework; 2) analysis of maximum attainable CPU temperatures for CPU frequency scaling states; 3) analysis of the impact of different temperature thresholds on CPU temperature and frequency; and 4) CPU temperature healing strategies.

The rest of this paper is organized into the following sections. Section II discusses the background and motivations of this study. Section III describes the general design of the automated healing of the overheated CPU. We discuss the reference implementation in section IV and present the results in section V. Section VI explains the related work, and section VII concludes this study and discusses possible further research.

II. Background and Motivations

Numerous studies have investigated the data center thermal design problem at different levels, including the chip level [10], server level [11], chassis level [3], rack level [12], and plenum level [3]. The objective of this study is to provide thermal control of the CPU at the chip level. Real-world experiences indicate that extraordinary compute-intensive workloads, cooling failures, or use of economizers can significantly increase the CPU temperature from a normal to a critical level. For example, at the High Performance Computing Center (HPCC) of Texas Tech University (TTU) [13], we experienced the following real-world scenarios where the proposed automated thermal control methodology can be applied.

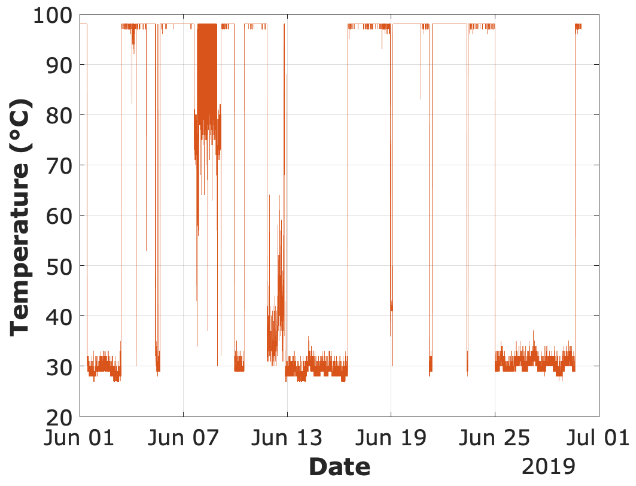


Fig. 1: CPU temperature analysis of the node "compute-6-2" over a month at the High Performance Computing Center of Texas Tech University.

A. *Effect of Extraordinary Compute-intensive Workloads*

Fig. 1 shows the CPU temperature analysis of a typical node in an HPC cluster. This time-series data was acquired at the sampling rate of 60 seconds, over a period of one month. As depicted in Fig. 1, the CPU temperature of the node compute-6-2 in the cluster frequently jumped to the critical temperature level. The observation indicated that continuous compute-intensive workloads are the major reason behind these fluctuations and driving CPU temperatures into the critical range. As this CPU was frequently running at a critical temperature over a long period of time, eventually the CPU completely stopped functioning. This finding shows that the average reflection plenum temperature is not always a reliable metric. The critical thermal state of an individual CPU can easily be ignored, which occurs due to extraordinary compute-intensive workloads. Therefore, it is important to track and control the CPU temperature at an individual node rather than rely solely on the data center room temperature. The techniques proposed in this research study effectively identify and heal an overheated CPU.

B. *Cooling Infrastructure Failure*

The occurrence of a cooling infrastructure (e.g., computer room air conditioning (CRAC) unit, chiller, CRAC power) failure can cause a sudden increase in the CPU temperature across the HPC cluster. This is another instance of a thermal related incident occurring at TTU's HPCC. At approximately 3:20 pm, March 21st, 2019, the chiller of the facility started delivering water at a higher than expected temperature, which led the CRAC units to blow insufficiently cooled air to remove the heat generated by the CPUs. As a result, the temperature of these CPUs started rising to higher than an acceptable level. We observed that this had little impact on CPU temperature of idle nodes; however, the CPUs with nodes running jobs had a drastic temperature increase. Fig. 2 shows

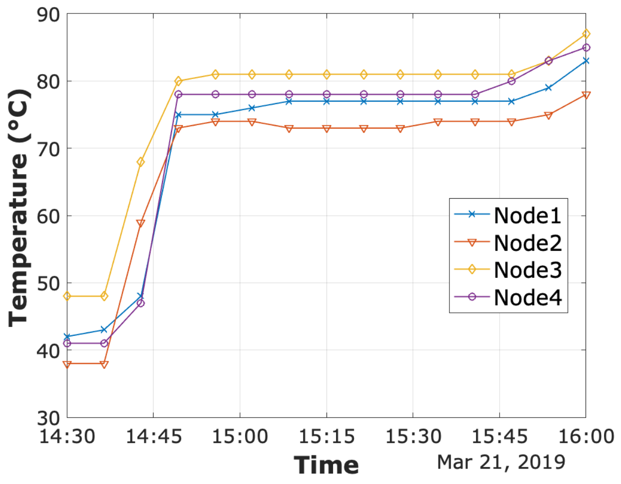


Fig. 2: Sudden increase of CPU temperature of some HPC nodes from the normal temperature to the critical level, due to the malfunction of the chiller.

the increase of the CPU temperatures of some HPC nodes from the normal to the critical level as a result of this incident. The proposed techniques in this study will prevent a sudden dramatic increase of the CPU temperature to the critical level and augment the ride-through time, which is the duration of the time before equipment starts going into thermal shutdown [6].

C. Usage of Economizers

Using economizers has a direct impact on improving power usage effectiveness (PUE) [14] and total-power usage effectiveness (TUE) [15]. Economized-based cooling uses up to 70% less energy [16]. However, due to economizers' strong dependency on unpredictable environmental conditions (e.g., seasonal variations including day and night temperature and humidity), economizer-based cooling is a common cause of thermal transients in HPC data centers. ASHRAE TC9.9 notes that normally the data center's ambient temperature fluctuates about 2°C (3.6°F) from the normal operating temperature in HVAC-cooled data centers [6]. By comparison, in an economizer-cooled data center, the ambient temperature of the data center could fluctuate between 18°C (64.4°F) and 27°C (80.6°F) from the normal operating temperature [6]. This wide range in operational thermal levels in an economized data center can be a source of a transient rise in the CPU's temperature. The proposed techniques are also helpful to control thermal conditions of CPUs in the HPC data center using economizers for cooling.

The above subsections describe representative scenarios that can have adverse impacts on the CPU's temperature and the resulting consequences. If the CPU thermal conditions are not automated and controlled appropriately, this may lead to a component loss and consumption of more power (e.g., more revolutions per minute (RPM) of CPU fans) in an effort to reduce the CPU temperature.

III. Automated Dynamic Thermal Control Design

The objectives of this study are to define a new level of instrumentation and deterministic methodologies for maintaining the CPU temperature to deliver desired performance, ensure infrastructure health, and consume energy efficiently in an autonomic manner. The design goals for the automated dynamic thermal control of the CPU include: 1) allowing the scaling up of the CPU performance when operating within a normal temperature range and 2) healing the temperature of the overheated CPU to be within its normal operating temperature range by scaling down CPU performance. We discuss the methodologies in detail below.

A. CPU Thermal Control

This study uses an existing framework, Kraken [17], [18], for CPU thermal control. Kraken is a distributed state discovery and control engine that can maintain many states across a large-scale computing cluster. The overall framework consists of the Kraken core and modules. The Kraken core provides a set of generic service engines that automate and control node states. The Kraken modules are a counterpart of the Kraken core and implement specific node management functions. The modules can perform functionalities, including the discovery of states, states' mutations, or both. By design, Kraken works in a parent and child manner (the term parent is synonymous to master). The Parent and Child Krakens communicate via a UDP protocol. They are identical and instances of the same codebase; however, various Kraken modules can be enabled and executed in the context of either parent or child. The Parent Kraken is deployed if it runs on a central node, whereas the Child Kraken is deployed if it runs locally on each node in the cluster. The state mutations or state discoveries can either run in the context of the Child or Parent Kraken, but not in the context of both simultaneously. To assist in understanding the proposed design, we further explain three key terms frequently used in Kraken-based design. First, a *configured state* refers to the configuration data, which describes the desired state of the node. The configuration state data is injected into Kraken at the deployment time. Second, a *discovery state* is an actual state of the node as monitored by Kraken in real-time. Third, a *state mutation* is a mechanism to control the state. It is a transition from one state to another, to achieve the configured (desired) state.

Kraken monitors CPU thermal and CPU frequency states and performs related mutations in a real-time manner using CPU thermal state discovery, CPU frequency scaling state discovery, CPU thermal state mutation, and CPU frequency scaling state mutation functions, which are described below.

B. CPU Thermal State Discovery

The CPU thermal state is an indicator of the intensity of CPU temperature. The CPU temperature is monitored in a real-time manner and the temperature is categorized as either `CPU_TEMP_NORMAL`, `CPU_TEMP_HIGH` or `CPU_TEMP_CRITICAL` thermal state, based on the lower and upper thresholds of those thermal states as shown in Fig. 3.

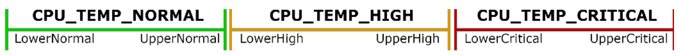


Fig. 3: Characterization of Thermal States

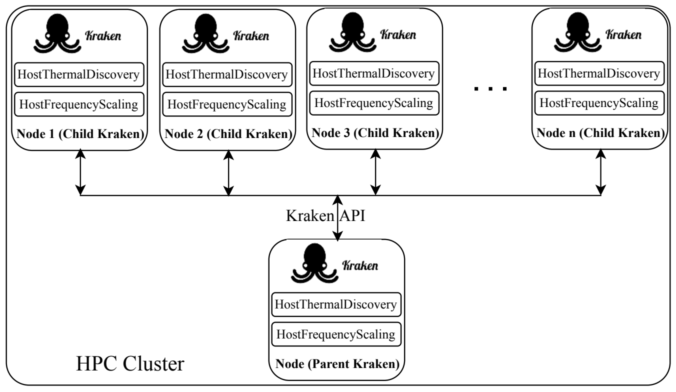


Fig. 4: In-band CPU Thermal Control Model

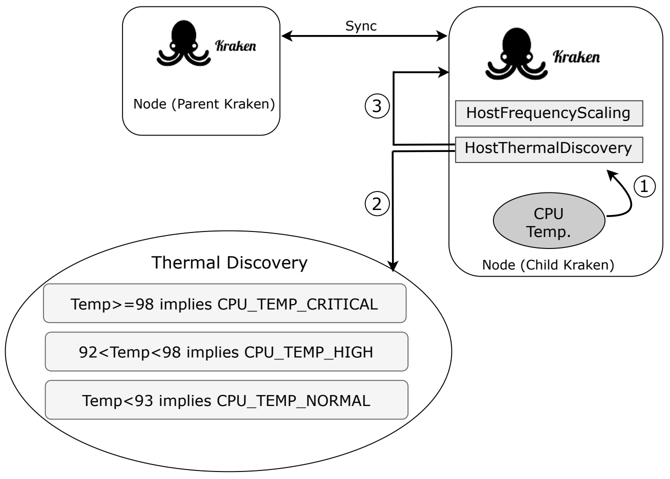


Fig. 5: In-band CPU Thermal Discovery

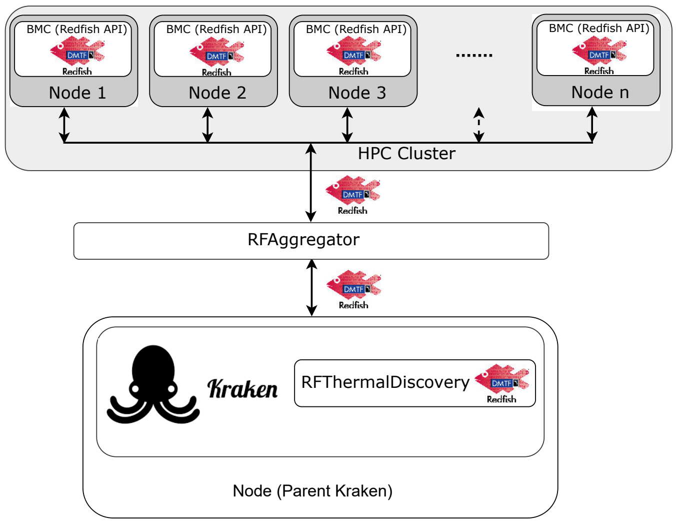


Fig. 6: Out-of-band Temperature Control Model

The temperature monitoring can be performed through in-band or out-of-band models.

1) In-band CPU Thermal State Discovery: An in-band CPU thermal state discovery involves accessing temperature information through the operating system (OS) of the node. The overall in-band thermal state discovery involves the Parent Kraken on a central node and the Child Kraken and the HostThermalDiscovery module [19] on each node as depicted in Fig. 4. The Child Kraken using the HostThermalDiscovery module discovers the CPU thermal state. Fig. 5 shows the CPU thermal state discovery process, which includes three major steps: 1) acquisition of CPU temperature, 2) determination of the CPU thermal state, and 3) generation of the thermal state discovery event to the Kraken core. The CPU temperature is acquired by reading the CPU thermal sensor using a generic OS routine or sensor-specific interface. The module keeps track of the previously acquired CPU temperature and performs steps 2 and 3 when the CPU temperature changes. Lastly, a thermal state discovery event is generated when the CPU temperature changes. The Child Kraken periodically synchronizes its thermal state with the Parent Kraken.

2) Out-of-Band CPU Thermal Discovery: An out-of-band (OOB) CPU thermal state discovery comprises accessing CPU temperature remotely via the baseboard management controller (BMC) without OS support. The OOB communication is performed through the BMC using the out-of-band protocols, such as the intelligent platform management interface (IPMI) [20] or the DMTF Redfish API [21]. The Redfish API is a RESTful standard interface used to manage the data center infrastructure. The API is specified in terms of a standard, machine-readable schema, with the payload of the messages being expressed in JavaScript Object Notation. Hojati, et al. [22] is a study based on the advantages of Redfish [23] to calculate energy efficiency and performance of equipment in data centers. We used Redfish API to perform OOB thermal discovery. The CPU thermal state discovery using the OOB paradigm requires RFThermalDiscovery module [19] and the Redfish Aggregator (RFAggregator) on the Parent Kraken. The RFThermalDiscovery module interacts with the remote Redfish-enabled BMCs to acquire CPU temperature using Redfish via the RFAggregator. The CPU temperature acquisition setup is shown in Fig. 6. The RFThermalDiscovery module in the Parent Kraken periodically initiates the Redfish group request via the RFAggregator for the acquisition of the CPU temperature from a group of BMCs. The RFAggregator fans out the Redfish API request to the individual Redfish-enabled BMCs. The BMC reads the CPU thermal sensor and sends the response to the RFAggregator. The RFAggregator then combines individual responses into a single aggregated response. The RFAggregator sends the aggregated response to the RFThermalDiscovery module in the Parent Kraken. The module compares the current CPU temperature with the previously acquired CPU temperature. As the CPU temperature changes, the module determines the CPU thermal state and generates the thermal state discovery event to the Kraken core.

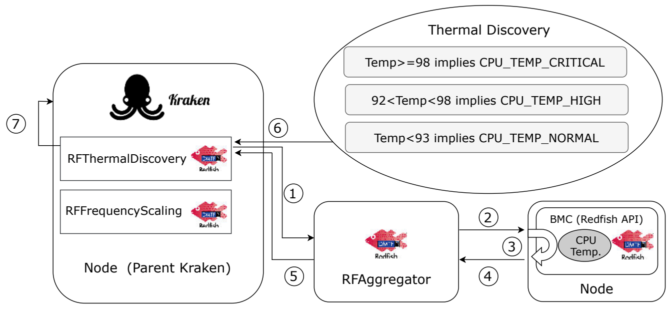


Fig. 7: Out-of-band CPU Thermal Discovery

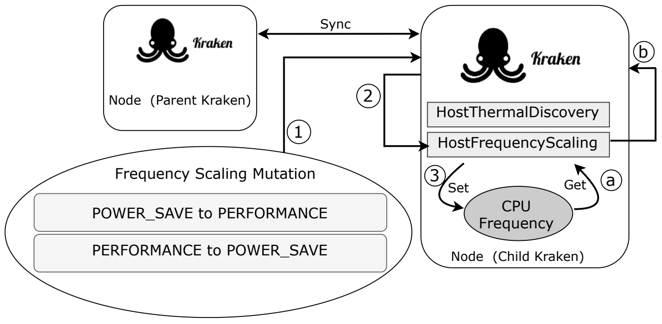


Fig. 8: CPU Frequency Scaling Procedure

Fig. 7 shows the major steps related to the discovery of the CPU thermal state via OOB.

C. CPU Frequency Scaling State Discovery

The CPU frequency scaling state corresponds to a generic CPU frequency scaling governor in the Linux *CPUFreq* subSystem [24]. In this study, we used `POWER_SAVE` and `PERFORMANCE` as CPU frequency scaling states. The `POWER_SAVE` and `PERFORMANCE` states correspond to the selection of the lowest CPU clock frequency and the highest CPU clock frequency, respectively. The Child Kraken discovers the CPU frequency scaling state of the node using the HostFrequencyScaling module [19] as shown in Fig. 8. The HostFrequencyScaling module periodically reads the current CPU frequency scaling state and compares it with the previous CPU frequency scaling state. In the event that the CPU frequency scaling state is changed, the module generates a CPU frequency scaling state discovery event to the Kraken core.

D. State Diagram of CPU Thermal Mutations

We designed a state diagram of Thermal Mutations. The CPU Thermal State Mutation allows Kraken to control CPU thermal state. When the CPU thermal state is `CPU_TEMP_HIGH` or `CPU_TEMP_CRITICAL`, CPU thermal state mutations provide a mechanism to revert to the desired CPU thermal state (i.e., `CPU_TEMP_NORMAL`). When a `CPU_TEMP_NORMAL` state is discovered, no thermal state mutation is performed. When the discovered thermal state is not the same as the desired thermal state, a corresponding mutation is performed. The possible CPU thermal state mutations are shown in Fig. 9.

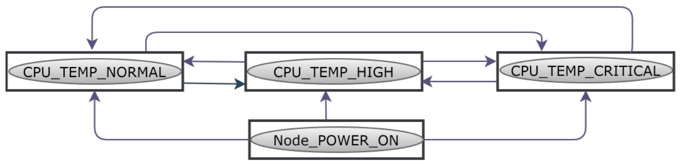


Fig. 9: State Diagram of CPU Thermal Mutations

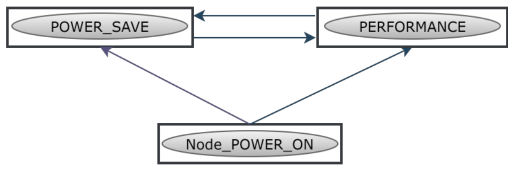


Fig. 10: State Diagram of CPU Frequency Scaling Mutations

E. State Diagram of CPU Frequency Scaling Mutations

We designed a state diagram of CPU Frequency Scaling Mutations. The CPU frequency scaling state mutations enable scaling up or down the CPU operating frequency. The CPU frequency scaling state can be mutated from `POWER_SAVE` to `PERFORMANCE` and vice versa. These mutations are governed by real-time CPU thermal states. Fig. 10 shows the possible CPU frequency scaling state mutations. The Child Kraken mutates CPU frequency scaling states locally using the HostFrequencyScaling module [19] as depicted in Fig. 8. When the Kraken core receives a CPU thermal state discovery event, a CPU frequency scaling state discovery event, or both, the Kraken core computes a CPU frequency scaling state mutation path corresponding to the received discovery event(s). The Kraken core signals the HostFrequencyScaling module to handle the mutation, which mutates the CPU frequency scaling state by changing the CPU frequency. The Child Kraken periodically synchronizes its state to the Parent Kraken.

IV. Reference Implementation

This section describes an example implementation of the automated dynamic thermal control design (section III). It illustrates the experimental setup, software implementation, and the Kraken dashboard that visualizes the cluster.

A. Experimental Setup

Hardware Setup: The hardware infrastructure includes a cluster of 150 Raspberry Pi3 nodes, one master node, and a fan. This arrangement is intended to emulate an HPC data center. Table I provides information related to the hardware involved in this implementation. **System Software:** The master node and Raspberry Pi3 nodes used CentOS 7.6 and *u-root*, respectively. *u-root* is a root file system, which packages firmware images along with Linux Kernel and other binaries as a single root binary [25].

TABLE I: Hardware Specifications

Master Node Specifications:	
CPU	2 x 4 cores AMD Opteron™ 6100, x GHz
RAM	23 GB DDR3
STORAGE	2TB HDD
NETWORK	1Gbit/s, Broadcom NetXtreme II
Raspberry Pi 3 Node Specifications:	
SoC	Broadcom BCM2837
CPU	4× ARM Cortex-A53, 1.2 GHz
RAM	1GB LPDDR2, 900 MHz
NETWORK	10/100M Ethernet

B. Software Stack

The Kraken and relevant Kraken modules are used in the realization of automated thermal control. **Kraken Framework:** It consists of the Parent Kraken and the Child Kraken, which are exactly the same binary image and identify themselves as either the parent or the child at the Kraken launch time. The Parent Kraken runs on the master node and the Pi3 nodes acquire the Child Kraken image (*u-root*) via Preboot Execution Environment (PXE). **Kraken Modules:** The implementation requires the following modules:

a) pipxe: Provides PXE-boot capabilities for Raspberry Pi3 nodes.
b) rfpipower: Emulates Redfish power control capabilities. Powers on and off the Raspberry Pi3 nodes remotely.
c) cpuburn: Induces load and stresses the Raspberry Pi3 nodes.
d) RFAggregator: Emulates and handles the Redfish group requests.
e) RFThermalDiscovery: Emulates the Redfish API to monitor CPU temperature of the Raspberry Pi3 nodes using RFAggregator.
f) HostThermalDiscovery: Performs CPU temperature monitoring of the Raspberry Pi3 nodes through the in-band mechanism. It reads temperature from a thermal sensor located at /sys/devices/virtual/thermal/thermal_zone0/temp/. The sections *HostThermalDiscovery Module Configuration* and *HostThermal Extension Configuration* in Table II describe the node and cluster level configurations, respectively.
g) HostFrequencyScaling: Makes use of DVFS to scale up or scale down CPU frequency to control CPU temperature and CPU performance. The HostFrequencyScaling module accesses different *CPUFreq* objects available at/sys/devices/system/cpu/cpufreq/policy0/. In particular, this module adjusts *scaling_governor*, *scaling_min_freq*, and *scaling_max_freq* objects to mitigate the CPU thermal and CPU performance requirements. Any change to these objects causes the selection of new frequency scaling governor. As a result, CPU is clocked to new frequency dynamically via platform specific CPU frequency driver (i.e. *cpufreq-cpu0*). The sections *HostFrequencyScaling Module Configuration* and *HostFrequencyScaler Extension Configuration* in Table III describe the node and cluster level configurations, respectively.

TABLE II: Thermal Configuration

HostThermalDiscovery Module Configuration:	
Parameter	Description
PollingInterval	sampling rate at which temperature is collected from Pi3 node. Its default value is 1 second.
TempSensorPath	CPU temperature reading location. In Pi3 node, CPU temperature is available at */sys/class/thermal/thermal_zone0/temp*.
LowerNormal	lower temperature threshold for the `CPU_TEMP_NORMAL` state. The default value is 3°C (37.4°F)
UpperNormal	upper temperature threshold for the `CPU_TEMP_NORMAL` state. The default value is 79°C (176°F)
LowerHigh	lower temperature threshold for the `CPU_TEMP_HIGH` state. The default value is 80°C (176°F)
UpperHigh	upper temperature threshold for the `CPU_TEMP_HIGH` state. The default value is 98°C (208.4°F)
LowerCritical	lower temperature threshold for the `CPU_TEMP_CRITICAL` state. The default value is 2°C (37.4°F)
UpperCritical	upper temperature threshold for the `CPU_TEMP_CRITICAL`. The default value is 99°C (208.4°F)
HostThermal Extension Configuration:	
State	This represents thermal state of CPU. It can be `CPU_TEMP_NORMAL`, `CPU_TEMP_HIGH`, or `CPU_TEMP_CRITICAL`. The desire (configuration) thermal state is `CPU_TEMP_NORMAL`.

C. Kraken Dashboard

The Kraken dashboard provides different states of the nodes in the cluster in realtime. Each box represents a node. The four triangles and border of a node represent states. Fig. 11 shows the master and states of 150 compute nodes. The border around each box and each of the four triangles within the box represent node states. The CPU frequency scaling, CPU temperature, PXE, runtime, and physical states and their possible values are shown in the legend section (left). As described in the legend, the border and upper triangle represent the CPU thermal state and CPU frequency scaling state, respectively. Overall, 139 nodes are in the desired thermal (i.e., `CPU_TEMP_NORMAL`) and frequency scaling (i.e., `PERFORMANCE`) states. One node is identified in an overheated thermal state (i.e., `CPU_TEMP_HIGH`) and is rightly switched to `POWER_SAVE` frequency scaling state for healing the CPU temperature. The remaining nodes are in the booting process.

V. Experimental Results

The results are described in three parts. First, maximum attainable CPU temperatures for each CPU frequency scaling state are analyzed. Second, the impact of scaling down CPU frequency at different temperature configuration thresholds for both CPU thermal and frequency scaling states are explored. Third, temperature- and time-based strategies for healing an overheated CPU is provided. All the following experiments were conducted with a CPU load of 100% at a room temperature of 30°C (86°F). An external fan was used to blow air (at room temperature) on the Raspberry Pi3 nodes.

TABLE III: CPU Frequency Scaling Configuration

HostFrequencyScaling Module Configuration:	
Parameter	Description
FreqSensorUrl	CPU frequency objects location. In Pi3 node, CPU frequency policy and attributes are available at */sys/devices/system/cpu/cpufreq/policy0/*.
HighToLowScaler	specifies which frequency scaling state to use to switch from high to low performance. Default value is powersave.
LowToHighScaler	specifies which frequency scaling state to use to switch from low to high performance. Default value is performance.
TimeBoundThrottle RetentionDuration	If TimeBoundThrottleRetention is activated, this parameter defines time duration (in minutes) to keep CPU in throttling mode.
ThrottleRetention	An option specifying whether or not to enforce throttle retention.
TimeBoundThrottle Retention	A boolean value that defines whether or not to enforce time-based throttling.
ThermalBoundThrottle Retention	A boolean value that defines whether or not to enforce throttling based on a certain temperature value.
ThermalBoundThrottle RetentionThreshold	If ThermalBoundThrottleRetention is activated, this parameter defines thermal threshold value (in Celsius).
ScalingGovernor	This parameter defines the CPU frequency scaler. Its default value is `POWER_SAVE`.
ScalingMinFreq	This parameter defines the minimum CPU frequency level (in MHz). Pi3 node minimum frequency value is 600 MHz.
ScalingMaxFreq	This parameter defines the maximum CPU frequency level. Pi3 node maximum frequency value is 1400 MHz.
HostFrequencyScaler Extension Configuration:	
States	This represents frequency scaling state of CPU. It can be `PERFORMANCE` or `POWER_SAVE`. The default configured state is `PERFORMANCE`.

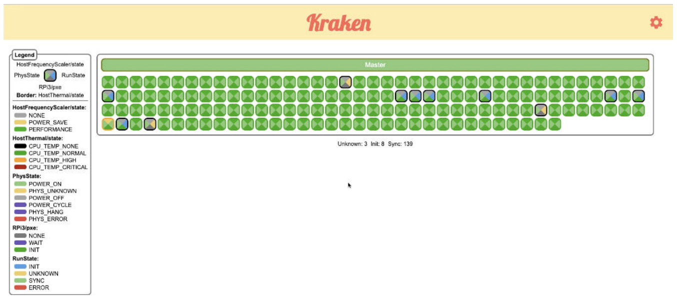


Fig. 11: Kraken dashboard

A. Maximum Attainable CPU Temperature for CPU Frequency Scaling States

To devise a better thermal control using CPU frequency scaling states, it is important to analyze and understand the maximum attainable CPU temperature for each frequency scaling state at room temperature. Fig. 12 depicts maximum CPU temperatures observed for `POWER_SAVE` and `PERFORMANCE` states. In the `POWER_SAVE` state, the CPU temperature increased up to 72°C (161.6°F) after ∼20 minutes, while in the `PERFORMANCE` state, it increased up to 86°C (186.8°F) after ∼10 minutes. This demonstrates that the CPU temperature can not be reduced below 72°C (161.6°F) using `POWER_SAVE` state when the CPU is running at 100% load.

B. Impact of CPU Temperature Thresholds on Temperature and Frequency

The objective of these analyses is to understand the impact of scaling down CPU frequency at different CPU temperature thresholds on CPU temperature and frequency. As described previously the CPU temperature can not be reduced below 72°C (161.6°F) when applying the `POWER_SAVE` state at room temperature. However, the CPU thermal behavior is not known when applying the `POWER_SAVE` state at different temperature thresholds. Fig. 13 (a)-(h) shows variations in CPU temperature and frequency when the CPU frequency is scaled down at different CPU temperature thresholds, namely, 50°C (122°F), 55°C (131°F), 60°C (140°F), 65°C (149°F), 70°C (158°F), 75°C (167°F), 80°C (176°F), and 85°C (185°F). These eight experiments were conducted to identify the best temperature threshold for scaling down the CPU frequency, which can provide more effective thermal control. Mathematically, the identified temperature threshold is the temperature where the CPU thermal resistance is minimal (see Eq. 1).

Overall, the results demonstrated in Fig. 13 (a) - (h) provide three patterns. First, when the CPU frequency scaling state was scaled down from `PERFORMANCE` to `POWER_SAVE` in the temperature threshold range of 50° to 65°C (122°F to 149°F), there were frequent switches between `POWER_SAVE` and `PERFORMANCE` for an initial short interval. In other words, when `POWER_SAVE` reduced the CPU temperature below the threshold temperature, the CPU frequency scaling state was changed to `PERFORMANCE` and when the CPU temperature crossed the threshold temperature, the CPU frequency scaling state was switched back to `POWER_SAVE`. After a while, the frequency scaling state remained in `POWER_SAVE` and the CPU temperature was accelerated to ∼72°C (161.6°F) i.e., maximum attained temperature for the `POWER_SAVE` state. Second, when the CPU frequency scaling state was scaled down in the range of 70° to 75°C (158°F to 167°F), there were frequent switches between `POWER_SAVE` and `PERFORMANCE`, and the temperature was ∼72°C (161.6°F) for the whole duration. Third, when the CPU frequency scaling state was scaled down in the range of 80° to 85°C (176°F to 185°F), unlike the other two patterns, there were fewer switches between `POWER_SAVE` and `PERFORMANCE`. The CPU frequency scaling state mostly remained in `PERFORMANCE`, and temperature increased beyond the thresholds (i.e., 80°C (176°F) or 85°C (185°F)) due to increased thermal resistance (i.e., temperature control became less effective).

Fig. 14 shows the summary of the scaling down of CPU frequency scaling state at eight different temperature thresholds. Our empirical results indicated that temperature in the range of 70° to 75°C (158° to 167°F) is the best CPU temperature threshold to scale down the CPU frequency to reduce the CPU temperature.

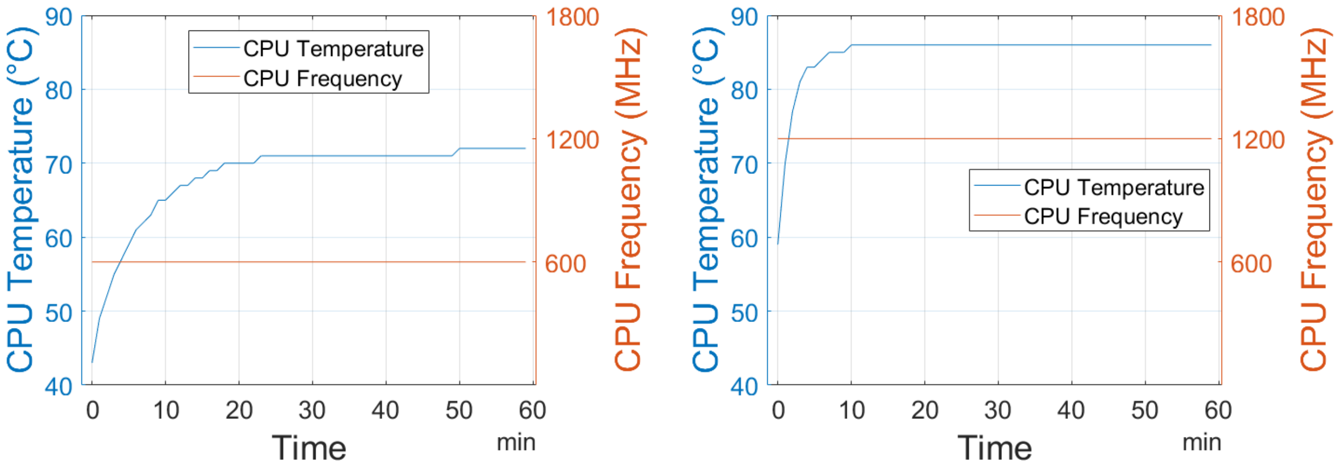


Fig. 12: Maximum attainable CPU temperatures for `POWERSAVE` (left) and `PERFORMANCE` (right) states.

VI. Related Work

The existing efforts related to CPU thermal control can be grouped into two areas: a) mechanical-based thermal control and b) software-based thermal control.

From the mechanical perspective, numerous cooling solutions have been studied, including air cooling [26], [27] and liquid cooling [28], [29]. These solutions are primarily designed to provide efficient and dependable inlet temperature control at the rack and room levels [30] by extricating the heat generated by power consumption. This indirectly reduces the CPU temperature without performance loss. While these mechanical cooling solutions can handle room and rack level temperature efficiently, they may not be responsive to possible thermal violations on an individual CPU due to a compute-intensive job.

Concerning the software-based thermal solutions, DVFS is the widely used technique to control the CPU temperature. This software-based component-level thermal control is helpful to handle the on-chip temperature. It allows the last-mile control of the thermal issues not handled by the mechanical level cooling solutions. Peluso et al. [31] analyzed the relationship between the CPU temperature and the hyper-parameters of the convolutional neural networks (CNN) using the DVFS knob. In comparison to their work, our method specifically addresses thermal violation for any CPU-intensive application. Kim et al. [32] provided a temperature-aware DVFS scheme for mobile devices. It estimates power and performance within the mobile device thermal budget. In contrast, the strategies proposed in our work effectively reduce the temperature of an overheated CPU.

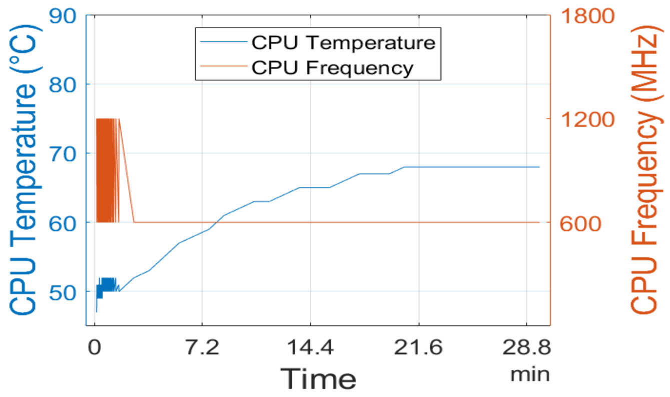
VII. Conclusions and Future Work

Often, modern CPUs are prone to overheating in unfavorable thermal conditions, including compute-intensive workloads on CPU, cooling infrastructure failures, and usage of economizers. Hence, an automated thermal state discovery and healing of overheated CPUs in the data center setting is highly desirable. This study has successfully shown the automated healing of the CPU temperature of a node. This implementation leverages state-of-the-art Kraken, which is an HPC automation and control framework. Our methodology discovers the CPU temperature using the DMTF Redfish API and exerts thermal control utilizing the DVFS mechanism. This methodology performs periodic discovery of the CPU thermal state. It reverts the CPU temperature within a normal temperature range of an overheated CPU by scaling down its CPU frequency. Our method allows scaling up the CPU frequency when the CPU thermal state is back to normal. This methodology heals the transient overheating of a CPU mainly caused by compute-intensive workloads or air economizers that cannot provide sufficient cooling (due to high outside air temperature). In case of CPU overheating due to fundamental failures (cooling equipment power failure or malfunctioning), this methodology increases ride-through time.

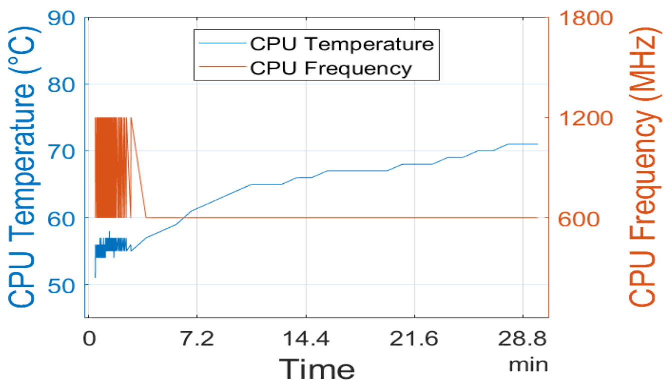
Further study and development are needed to cover more thermal control use cases. First, more investigation is needed to scale down the CPU frequency coherently to heal a node involved in a group of nodes executing a message passing interface (MPI) job. Second, we plan to perform a graceful powering-off of a node when the CPU thermal state reaches an irreversible critical level. Third, more study is needed to integrate the proposed thermal control with other data center infrastructure management solutions.

Acknowledgment

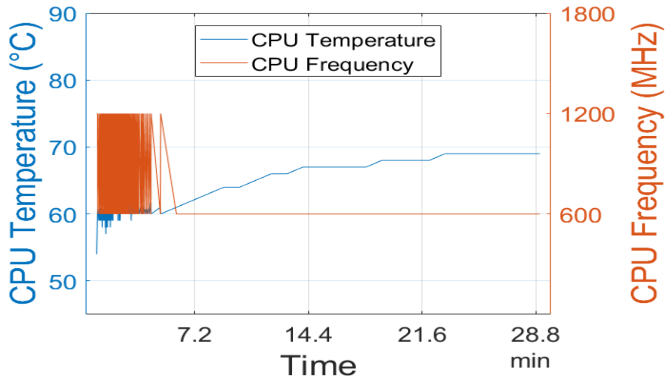
This manuscript has been approved for unlimited release and has been assigned LA-UR-21-25032. This work was supported by the U.S. Department of Energy through the Los Alamos National Laboratory. Los Alamos National Laboratory is operated by Triad National Security, LLC, for the National Nuclear Security Administration of U.S. Department of Energy (Contract No. 89233218CNA000001). This work was also supported by the Ultrascale Systems Research Center which is a collaboration between the New Mexico Consortium



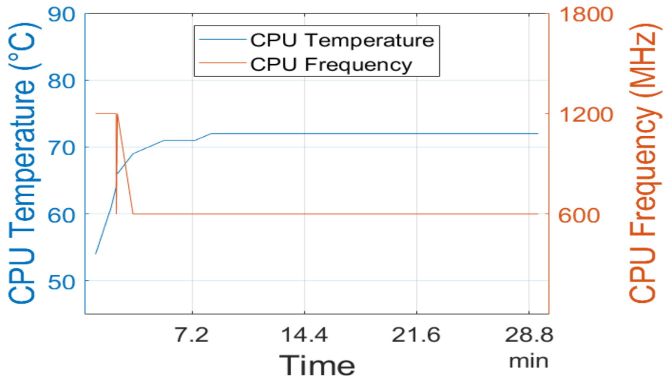
(a) `POWER_SAVE` at 50°C.



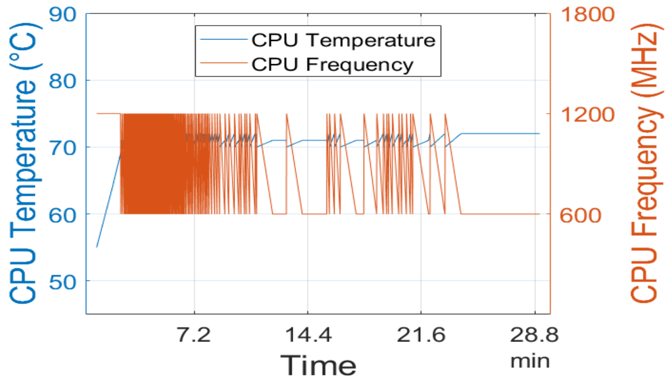
(b) `POWER_SAVE` at 55°C.



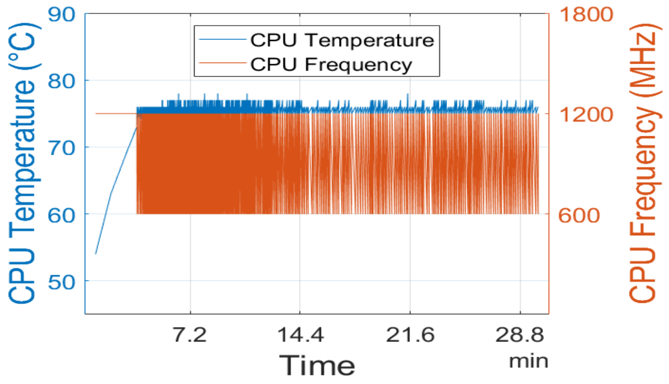
(c) `POWER_SAVE` at 60°C.



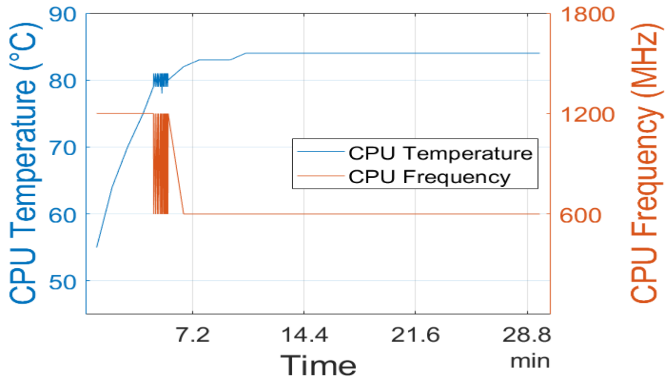
(d) `POWER_SAVE` at 65°C.



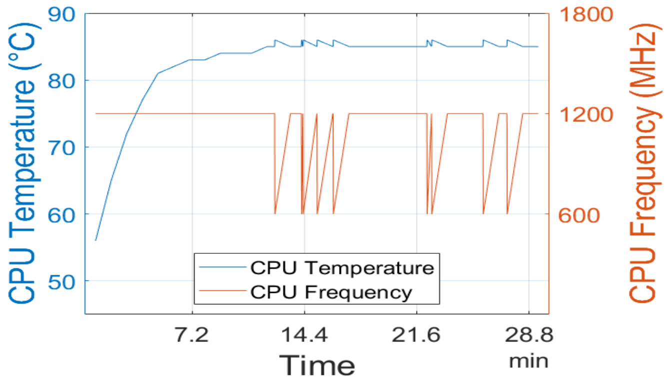
(e) `POWER_SAVE` at 70°C.



(f) `POWER_SAVE` at 75°C.



(g) `POWER_SAVE` at 80°C.



(h) `POWER_SAVE` at 85°C.

Fig. 13: Variations in CPU temperature and frequency at different CPU temperature thresholds, namely, 50°C (122°F), 55°C (131°F), 60°C (140°F), 65°C (149°F), 70°C (158°F), 75°C (167°F), 80°C (176°F), and 85°C (185°F).

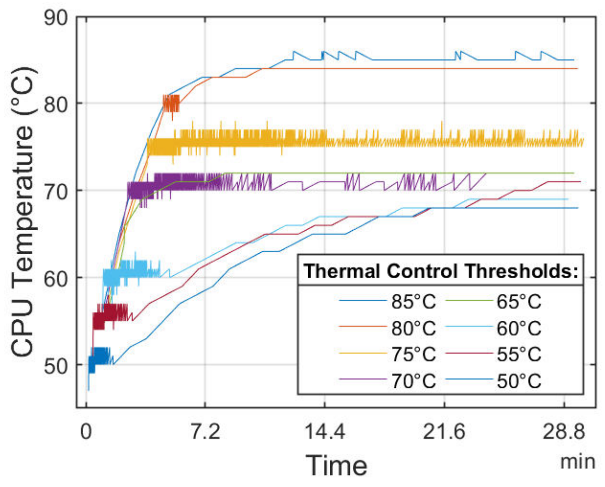


Fig. 14: Summary of scaling down of CPU frequency scaling state at eight different temperature thresholds.

and the Los Alamos National Laboratory. This research was supported in part by the National Science Foundation under grant CNS-1939140 (A U.S. National Science Foundation Industry-University Cooperative Research Center on Cloud and Autonomic Computing) and OAC-1835892. The authors are grateful to Cory Donald Lueninghoener, Kevin Pelzel, Elham Hajati and Alan Sill for reviewing and assisting this work.

References

[1] M. B. Taylor, "A Landscape of The New Dark Silicon Design Regime," *IEEE Micro*, vol. 33, no. 5, pp. 8–19, 2013.

[2] S. Borkar, "Thousand Core Chips: a Technology Perspective," in *Proceedings of the 44th annual design automation conference*, 2007, pp. 746–749.

[3] Y. Joshi and P. Kumar, *Energy Efficient Thermal Management of Data Centers*. Springer Science & Business Media, 2012.

[4] R. Viswanath *et al.*, "Thermal Performance Challenges from Silicon to Systems," 2000.

[5] R. A. Bridges, N. Imam, and T. M. Mintz, "Understanding GPU Power: A Survey of Profiling, Modeling, and Simulation Methods," *ACM Computing Surveys (CSUR)*, vol. 49, no. 3, pp. 1–27, 2016.

[6] ASHRAE, "Data Center Power Equipment Thermal Guidelines and Best Practices," 2016, https://tc0909.ashraetcs.org/documents/ASHRAE_TC0909_Power_White_Paper_22_June_2016_REVISED.pdf.

[7] Y. Kodama, S. Itoh, T. Shimizu, S. Sekiguchi, H. Nakamura, and N. Mori, "Imbalance of CPU Temperatures in a Blade System and its Impact for Power Consumption of Fans," *Cluster computing*, vol. 16, no. 1, pp. 27–37, 2013.

[8] J. Crapse, N. Pappireddi, M. Gupta, S. Y. Shvartsman, E. Wieschaus, and M. Wühr, "Evaluating the Arrhenius Equation for Developmental Processes," *Molecular Systems Biology*, vol. 17, no. 8, p. e9895, 2021.

[9] P. Lall, "Tutorial: Temperature as an Input to Microelectronics-Reliability Models," *IEEE Transactions on Reliability*, vol. 45, no. 1, pp. 3–9, 1996.

[10] X. Wei, Y. Joshi, and M. K. Patterson, "Experimental and Numerical Study of a Stacked Microchannel Heat Sink for Liquid Cooling of Microelectronic Devices," 2007.

[11] S. P. Gurrum, S. K. Suman, Y. K. Joshi, and A. G. Fedorov, "Thermal Issues in Next-Generation Integrated Circuits," *IEEE Transactions on device and materials reliability*, vol. 4, no. 4, pp. 709–714, 2004.

[12] M. K. Herrlin *et al.*, "Rack Cooling Effectiveness in Data Centers and Telecom Central Offices: The Rack Cooling Index (RCI)," *Transactions-American Society of Heating Refrigerating and Air conditioning Engineers*, vol. 111, no. 2, p. 725, 2005.

[13] HPCC. (2020) High Performance Computing Center. [Online]. Available: http:www.depts.ttu.edu/hpcc/

[14] V. Avelar, D. Azevedo, A. French, and E. N. Power, "PUE: a Comprehensive Examination of the Metric," *White paper*, vol. 49, 2012.

[15] M. K. Patterson, S. W. Poole, C.-H. Hsu, D. Maxwell, W. Tschudi, H. Coles, D. J. Martinez, and N. Bates, "TUE, a New Energy-Efficiency Metric Applied at ORNL's Jaguar," in *International Supercomputing Conference*. Springer, 2013, pp. 372–382.

[16] A. Kumar and Y. Joshi, "Use of Airside Economizer for Data Center Thermal Management," in *2008 Second International Conference on Thermal Issues in Emerging Technologies*. IEEE, 2008, pp. 115–124.

[17] J. Lowell Wofford, "Designing a Scalable Framework for Declarative Automation on Distributed Systems," *arXiv e-prints*, pp. arXiv–2104, 2021.

[18] "HPC Kraken," 2020, https://github.com/hpc/kraken.

[19] Kraken-Hpc, "Kraken HPC Legacy Modules." [Online]. Available: https://github.com/kraken-hpc/kraken-legacy/tree/main/modules

[20] "IPMI Specification, V2.0, Rev. 1.1: Document." [Online]. Available: https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html

[21] DMTF. (2020) DMTF's Redfish®. [Online]. Available: https://www.dmtf.org/standards/redfish

[22] E. Hojati, J. Hass, A. Sill, and Y. Chen, "Redfish Green500 Benchmarker (RGB): Towards Automation of the Green500 Process for Data Centers," in *2020 IEEE Green Technologies Conference(GreenTech)*, 2020, pp. 47–52.

[23] E. Hojati, Y. Chen, and A. a. Sill, "Benchmarking Automated Hardware Management Technologies for Modern Data Centers and Cloud Environments," in *UCC '17: Proceedings of the 10th International Conference on Utility and Cloud ComputingDecember 2017*, 2017, p. 195–196.

[24] "CPU Performance Scaling." [Online]. Available: https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html

[25] Kraken-Hpc-uroot, "Kraken HPC u-root." [Online]. Available: https://github.com/kraken-hpc/u-root

[26] J. Wan *et al.*, "Air Flow Measurement and Management for Improving Cooling and Energy Efficiency in Raised-Floor Data Centers: A Survey," *IEEE Access*, vol. 6, pp. 48 867–48 901, 2018.

[27] S. V. Patankar, "Airflow and Cooling in a Data Center," *Journal of Heat transfer*, vol. 132, no. 7, 2010.

[28] J. Gullbrand, M. J. Luckeroth, M. E. Sprenger, and C. Winkel, "Liquid Cooling of Compute System," *Journal of Electronic Packaging*, vol. 141, no. 1, 2019.

[29] H. B. Jang, I. Yoon, C. H. Kim, S. Shin, and S. W. Chung, "The Impact of Liquid Cooling on 3D Multi-Core Processors," in *2009 IEEE International Conference on Computer Design*. IEEE, 2009, pp. 472–478.

[30] J. Athavale, M. Yoda, and Y. Joshi, "Thermal Modeling of Data Centers for Control and Energy Usage Optimization," in *Advances in Heat Transfer*. Elsevier, 2018, vol. 50, pp. 123–186.

[31] V. Peluso, R. G. Rizzo, and A. Calimera, "Performance Profiling of Embedded Convnets Under Thermal-Aware DVFS," *Electronics*, vol. 8, no. 12, p. 1423, 2019.

[32] J. M. Kim, Y. G. Kim, and S. W. Chung, "Stabilizing CPU Frequency and Voltage for Temperature-Aware DVFS in Mobile Devices," *IEEE Transactions on Computers*, vol. 64, no. 1, pp. 286–292, 2013.