CoMETC: Coordinated Management of Energy/Thermal/Cooling in Servers

RAID AYOUB, Intel Corporation
RAJIB NATH and TAJANA SIMUNIC ROSING, University of California, San Diego

We introduce a Coordinated Management of Energy, Thermal, and Cooling (CoMETC) technique to minimize cooling and memory energy of server machines. State-of-the-art solutions decouple the optimization of cooling energy costs and energy consumption of CPU and memory subsystems. This results in suboptimal solutions due to thermal dependencies between CPU and memory and the nonlinearity in energy costs of cooling. In contrast, we develop a unified solution that integrates energy, thermal, and cooling management for CPU and memory subsystems to maximize energy savings. CoMETC reduces the operational energy of the memory by clustering active memory pages to a subset of memory modules while accounting for thermal and cooling aspects. At the same time, CoMETC removes hotspots between and within the CPU sockets and reduces the effects of thermal coupling with memory in order to minimize cooling energy costs. We design CoMETC using a control-theoretic approach to guarantee meeting these objectives. We introduce a formal thermal and cooling model to be used for online decisions inside CoMETC. Our experimental results show that CoMETC achieves average cooling and memory energy savings of 58% compared to state-of-the-art techniques at a performance overhead of less than 0.3%.

Categories and Subject Descriptors: B.8.2 [**Performance and Reliability**]: Performance Analysis and Design Aids

General Terms: Design, Algorithms, Measurement, Performance

ACM Reference Format:
Ayoub, R., Nath, R., and Rosing, T. S. 2013. CoMETC: Coordinated management of energy/thermal/cooling in servers. ACM Trans. Des. Autom. Electron. Syst. 19, 1, Article 1 (December 2013), 28 pages.
DOI: http://dx.doi.org/10.1145/2534381

1. INTRODUCTION

In recent years, server designers are striving to keep up with the unprecedented growth in computational demand. It has become the norm in server machines to have multiple CPU sockets and large DRAM memory to handle vast computations. The side effect of using large numbers of computational resources is the increase in power density in the system. High power dissipation elevates the operational costs of machines. It also causes thermal hotspots that have substantial effects on reliability, performance, and leakage power [Pedram and Nazarian 2006; Ajami et al. 2005]. Dissipating the excess heat is a big challenge as it requires complex and energy-hungry cooling subsystems.

A preliminary version of this article appeared in *Proceedings of the IEEE 18th International Symposium on High Performance Computer Architecture (HPCA'12)* [Ayoub et al. 2012].
This work has been funded by NSF grant numbers 0916127, 1218666, 1029783, and 812072; CNS, Oracle, Google, Microsoft, MuSyC, UC Micro grant 08-039, and Cisco.
Authors' addresses: R. Ayoub (corresponding author), Strategic CAD Labs, Intel Corporation; email: raid.ayoub@intel.com; R. Nath and T. S. Rosing, Department of Computer Science, University of California, San Diego, CA.

DOI: http://dx.doi.org/10.1145/2534381

Enhancing the energy proportionality of server machines requires improvement in the energy efficiency of the subsystems, particularity those that are power hungry. Traditionally, the CPU is known to be the primary source of system power consumption. This motivated designers over the years to enhance the energy efficiency of the CPU subsystem. Less attention was given to energy optimization of the rest of the system, which led to poor energy proportionality at the system level [Barroso and Holzle 2009]. The memory subsystem is the other major power-hungry component, as it consumes up to 35% of total system energy and has poor energy proportionality [Barroso and Holzle 2007, 2009]. The capacity and bandwidth of the memory subsystem are typically designed to handle worst-case scenarios. Applications may vary significantly in terms of their memory access pattern and memory footprint. One solution to improving energy proportionality is to activate a subset of the memory modules that are sufficient to serve the application's needs [Hai et al. 2005]. Such clustering, however, increases the power density of the active memory modules which could cause thermal problems.

Efficient management of thermal problems is another growing challenge. Thermal problems are not limited to CPU but also to memory, as both have high power density. To manage the high temperature in a CPU subsystem, a number of core-level dynamic thermal management (DTM) techniques have been proposed [Coskun et al. 2008; Yeo et al. 2008; Ayoub and Rosing 2009; Ayoub et al. 2011]. The scope of these techniques are limited to CPUs only and cannot mitigate thermal emergencies in memory. Lin et al. [2007, 2008] propose solutions to mitigate thermal emergencies in the memory system by throttling the throughput of memory to keep the temperature within the safe zone. However, these solutions do not improve the energy proportionality in the memory subsystem, as they do not consider minimizing the number of active DIMMs to just what is needed. In addition, the cooling energy costs are not considered in these techniques.

The common approach for removing the excess of heat from servers is to incorporate a fan subsystem. However, the operational costs of the fan is substantial, since the power consumed by a fan is cubically related to the air-flow rate [Patterson 2008]. The fan system in high-end servers consumes as much as 80 W in 1U rack servers and 240 W or more in 2U rack servers. Due to cost and area constraints, a common set of fans is normally used to cool both the CPUs and memory.[1] For such scenarios, the inlet temperature of the components at the end of the air-flow path (downstream) becomes a function of the temperature of the components located at the beginning of the air-flow path (upstream) in addition to the server's inlet temperature. Poor thermal distribution deviates the cooling subsystem from the energy-efficient operating point. These problems can be alleviated via thermal- and cooling-aware workload scheduling in the system.

In this work, we present CoMETC, a Coordinated Management of Energy, Thermal, and Cooling technique for servers which improves the Server Power Usage Efficiency (SPUE). Providing an integrated solution is necessary due to the thermal dependencies between the CPU and memory when both share the same cooling resources. CoMETC maximizes energy efficiency in the machine by controlling the number of active memory modules to just what is needed to provide sufficient storage capacity and bandwidth while minimizing operational energy. CoMETC also schedules the workload between the CPU sockets to create a balanced thermal distribution between them, not only to minimize the thermal coupling effect but also to mitigate their thermal hotspots as well. We developed a control-theoretic approach that controls memory page assignment (memory clustering), socket-level scheduling, and fan speed to guarantee convergence to the desired objectives. Finally, we show that applying CoMETC results in

[1] `http://www.intel.com/products/server/motherboards/s5400sf/s5400sf-overview.htm`.

58% average energy reduction of (memory and cooling) subsystems at a negligible performance overhead.

2. RELATED WORK

Energy and thermal management of memory and CPU subsystems is a vibrant area of research. Research in this domain can be classified broadly into four main categories: CPU thermal management, power management of memory subsystem, thermal management of memory subsystem, and cooling management.

CPU Thermal Management. In the past few years, several techniques have been proposed to mitigate the temperature problems at the core level. These techniques can be broadly classified into reactive and proactive techniques. Skadron et al. [2003] propose two reactive DTM techniques that manage heat aggressively. The first technique is based on dynamic voltage-frequency scaling, while the other uses pipeline throttling. In Choi et al. [2007], Heo et al. [2003], and Donald and Martonosi [2006], propose activity migration to manage high temperature by moving computations across replicated units when temperature reaches emergency levels. However, the reactive techniques can cause high performance overhead, and they are not so effective in improving thermal distribution across the chip. To overcome these problems, a class of proactive thermal management techniques are suggested to manage thermal problems ahead of time. Coskun et al. [2008] propose an autoregressive moving average (ARMA) model that is based on the serial autocorrelation in the temperature time-series data. The model is updated dynamically to adapt to possible workload changes. Although ARMA is fairly accurate, it requires a training phase that may degrade performance. Ayoub and Rosing [2009] suggest a new thermal predictor that is accurate and does not require runtime adaptations. This predictor utilizes the bandlimited property in the temperature signal. The authors implement this thermal predictor within a thermal management technique. The reported results show an appreciable reduction in occurrence of thermal hotspots. However, the optimizations in these techniques are limited to a single CPU socket, and they do not consider the cooling dynamics. Ayoub et al. [2011] propose a hierarchical thermal management that optimizes temperature across and within individual CPU sockets. This work accounts for cooling dynamics while managing temperature, which results in decent savings in cooling energy. Nevertheless, this technique does not address the energy and thermal challenges in memory subsystems.

Power Management for Memory Subsystem. A number of techniques have been proposed to reduce energy consumption in memory subsystems. Fan et al. [2001] optimize for DRAM power by placing memory in a low power state during idle periods. This approach is effective when there are frequent idle periods. Hai et al. [2005] mitigate memory power by clustering heavily-accessed pages to a subset of the memory modules, in particular, dual-in-line memory modules (DIMMs), and putting the rest of modules in a self-refresh mode. However, such consolidation increases power density that can cause thermal problems. These techniques do not address any thermal issues.

Thermal Management for Memory Subsystem. Thermal management techniques for memory subsystems have been proposed recently. Lin et al. [2007] handle thermal emergencies in memory by controlling memory throughput. The main drawback with this approach is the associated performance overhead. Lin et al. [2009], manage high temperature in memory by grouping the jobs so each group is mapped to a subset of DIMMs, assuming that all jobs in a group can run simultaneously. Only one group of the DIMMs is active at any point in time, while the rest stay inactive to cool down and save energy. The authors assume the bandwidth of each thread is known in advance, which is not a realistic assumption for general-purpose systems. Song et al. [2011] propose

a thermal management technique for memory subsystems at the microarchitectural level. However, this technique does not optimize for energy. All these techniques do not optimize for cooling energy, which is significant.

Cooling Management. A set of cooling management techniques is proposed to remove excess heat from the machines. In Wang et al. [2009] and Chiueh et al. [2000] propose an optimized closed-loop fan control. Wang et al. [2009] suggest an optimal fan speed algorithm for blade servers based on convex optimizations. As leakage depends on temperature, Shin et al. [2009] propose a fan control mechanism that optimizes for leakage power. A class of techniques is proposed to handle the cooling energy costs via a better workload scheduling. Heath et al. [2006] implement a thermal-aware workload balancing technique for server machines which uses component utilization as a proxy for thermal stress. Tolia et al. [2009] proposed a cooling-aware workload management technique to mitigate the energy costs in blade servers. A set of techniques has been suggested to improve cooling efficiency in data centers [Tang et al. 2007; Schmidt et al. 2005; Wei et al. 2011]. Tang et al. [2007] and Schmidt et al. [2005] introduce workload scheduling techniques to alleviate the air circulation problem around the racks of data centers. Wei et al. [2011] propose a technique that manages the cooling rate based on the utilization level of cooling zones. Nevertheless, these data center techniques are not so effective for mitigating the temperature and cooling energy costs within server machines. Patterson [2008] addresses the modeling of convective thermal resistance.

In this work, we make the following contributions.

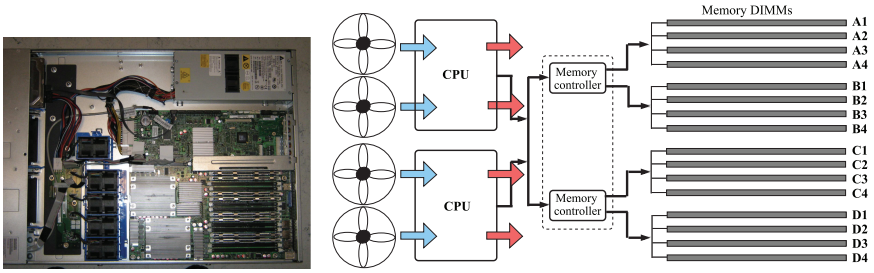
—We introduce CoMETC, a novel approach that unifies the management of energy, thermal, and cooling for CPU and memory subsystems using a control-theoretic framework to deliver high energy savings and stability of control. To the best of our knowledge, CoMETC is a first work that achieves these goals.
—We propose a new integrated thermal model for CPU and memory subsystems that considers the cooling dynamics. We have validated our model using measurements on a real machine.
—We report a detailed evaluation and discussion of our technique which delivers an average energy savings of 58% at a performance overhead of less than 0.3%.

3. INTEGRATED THERMAL AND COOLING MODEL FOR CPU AND MEMORY

In this section, we focus on developing an integrated thermal and cooling model for both CPU and memory which accounts for thermal dependency between them. We then investigate the opportunities of energy savings in memory and the associated thermal challenges.

Before we dive into details, we describe briefly the server we use to collect real-life measurements. Figure 1(a) shows a photo of our server (Intel Quad-Core dual-socket Xeon E5440), which is an illustrative example of a modern server. Each CPU is associated with a separate set of fans, where each cools both the CPU and subset of memory DIMMs, the CPU is placed close to the fan while the memory is downstream. Having a common cooling creates a thermal dependency between the upstream and downstream components that needs to be accounted for.

The memory subsystem in our server is shared among the CPU sockets where memory coherency is enforced by the hardware. This machine has two off-chip memory controller, where each is connected to memory by two memory channels, each channel is connected to four DIMM slots, as shown in Figure 1(b) (we use 4GB DDR2 DIMMs). To measure the power of each DIMM, we add an extender that has current sensors in the supply lines, where these sensors are connected to a data acquisition system. Benchmarks from the SPEC2000 suite have been used as workloads. The CPU, memory, and cooling specs are provided in a later table.



(a) Server machine. (b) System organization and cooling.

Fig. 1. Intel dual-socket Xeon server.

3.1. Air-flow Cooling Background

In general, both thermals and cooling can be modeled based on the known duality between electricity and temperature [Skadron et al. 2004]. Air cooling is commonly modeled via a convective thermal resistance, where its value is a function of the air-flow rate [Patterson 2008]. The value of this convective resistance, R_{conv}, can be computed as

$$R_{conv} \propto \frac{1}{AV^{\alpha}}, \tag{1}$$

where A is the effective area of the heat sink, V is the air-flow rate, and α is a factor with a range of (0.8–1.0). For a given heat flow, the smaller the value of R_{conv}, the lower the temperature of the component. To compute the cooling energy costs, we use the results from Patterson [2008] to relate the fan speed, F, with the air-flow rate as $V \propto F$. The cooling costs for changing the air-flow rate from V_1 to V_2 can be computed as [Patterson 2008; Shin et al. 2009]

$$\frac{P_2}{P_1} = \left(\frac{V_2}{V_1}\right)^3, \tag{2}$$

where P_1 and P_2 represent the fan's power dissipation at V_1 and V_2, respectively. For a given system with multiple fans, the Pareto point of fan speed to save power occurs when all fans spin at same speed due to the cubic relation between fan speed and its power. This indicates that having the fans spin at the optimal point can save significant energy.

3.2. System Thermal and Cooling Model

The inlet temperature of air flow to memory modules depends on the temperature of air flow at the entry point to the upstream CPUs and the exerted heat from the CPUs to the common air flow. However, the inlet temperature of air flow to the DIMMs may not be uniform, as the temperatures of upstream CPUs vary depending on their workload. In order to model the thermal dependency between CPUs and memory, we use dependent heat sources which model the extra heat generated from the upstream CPUs. Figure 2 shows the unified thermal/cooling model of the CPU and memory. Definitions of CPU and memory thermal model are discussed in Sections 3.2.1 and 3.2.2, respectively. The dependent coupling heat source of the memory, q^D, is proportional to the heat sink temperature of the CPU, T^C_{ha}, and inversely proportional to the case to ambient thermal resistance of the downstream memory modules, R^D_{ca}, as follows.

$$q^D \propto \frac{T^C_{ha}}{R^D_{ca}}. \tag{3}$$

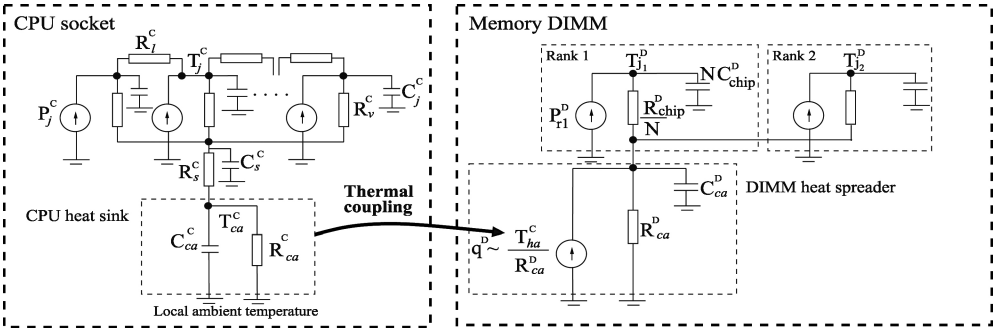


Fig. 2. Combined thermal model.

Increasing fan speed reduces the temperature of memory modules in two ways. First, it brings down the temperature of the upstream CPU heat sink, which reduces the coupling heat, q^D. In addition, higher fan speed reduces the convective thermal resistance of the memory modules, which further lowers the DIMMs temperature.

3.2.1. CPU Thermal Model. Figure 2 (left part) depicts the thermal model of the CPU chip with a thermal package which is based on Ayoub et al. [2011] and Skadron et al. [2004]. The thermal model of the CPU includes the die and the heat spreader. The steady state heat transfer in the vertical direction of the die is modeled via a vertical thermal resistance per component, R_v^C. Similarly, the steady state lateral heat transfer between the die's components is modeled using lateral thermal resistances, R_l^C. However, the lateral effect can be neglected at the core level [Heo et al. 2003]. The transient thermal behavior is modeled by including a thermal capacitance per component. The value of C_j^C represents the thermal capacitance of the components in the die. The power dissipation of each component is modeled as a heat source, P_j^C. For heat spreader, R_s^C and C_s^C refer to its associated thermal resistance and capacitance, respectively.

Heat sink is commonly used to enhance heat transfer between the CPU and the ambient inside the server. The heat flow from the CPU case to local ambient is modeled as a combination of conduction and convection heat transfers [Skadron et al. 2004]. The heat sink is assumed to be an isothermal conductive layer due to its high thermal conductance [Skadron et al. 2004]. We use R_{hs}^C to represent the value of the conductive thermal resistance of the heat sink. The convective heat flow is modeled by a convective resistance, R_{conv}^C, which is connected in series with R_{hs}^C, where their sum represents the *case to ambient* thermal resistance, R_{ca}^C ($R_{ca}^C = R_{hs}^C + R_{conv}^C$). The component R_{ca}^C is connected in parallel with the thermal capacitance of the heat sink, C_{ca}^C, which forms a single node RC circuit. The value of R_{conv}^C is calculated as a function of air flow using Eq. (1). We use T_j^C and T_{ca}^C to represent the core junction to temperature and the case to ambient temperature, respectively. The instantaneous value of the case to ambient temperature, $T_{ca}^C(t)$, can be expressed as

$$\frac{dT_{ca}^C(t)}{dt} = -\frac{T_{ca}^C(t)}{\tau_{ca}^C} + \frac{P^C(t)}{C_{ca}^C}, \tag{4}$$

where, τ_{ca}^C is the heat sink *time constant*, $\tau_{ca}^C = R_{ca}^C C_{ca}^C$. The $P^C(t)$ represents the instantaneous total power dissipation in the CPU socket.

3.2.2. Memory Thermal and Cooling Model. The DRAM subsystem is organized as an array of DIMMs (see Figure 1(b)), where each DIMM is composed of ranks, usually two, and

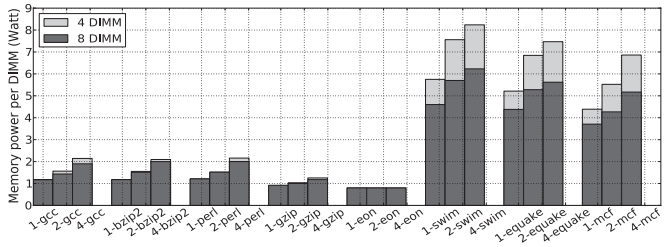


Fig. 3. Memory power per DIMM. The number in front of the benchmark is the quantity of instances we run.

each rank contains a number of memory chips (e.g., eight chips per rank). Each DRAM chip stores data corresponding to a subset of predefined bits of the data word (e.g., the first eight bits of a 64-bit data word maps to the first chip). This organization leads to a uniform distribution in power across the DRAM chips that belong to the same rank.

Figure 2 (right side) shows the thermal model of a single DIMM with a heat spreader. Superposition theory is used to simplify the RC network of the DRAM chips per rank to a single RC node. In Figure 2, the heat source, P^D_{ri}, represents the power dissipation in rank i, R^D_{chip} is the vertical thermal resistance of each chip, C^D_{chip} is the thermal capacitance of each chip, and T^D_j is the junction temperature of a DRAM chip. The number of DRAM chips in each rank and the number of ranks in each DIMM are assumed to be N and n_r, respectively.

The DIMMs are commonly equipped with a heat spreader to better dissipate their temperature. The heat spreader is modeled as a single node RC circuit, as shown in Figure 2, due its small horizontal thermal resistance relative to convective resistance [Lin et al. 2007, 2009]. The components C^D_{ca} and R^D_{ca} correspond to the thermal capacitance and *case to ambient* thermal resistance of the heat spreader, respectively.

The transient behavior of the junction temperature is dominated by the heat spreader temperature dynamics over the long run as the DRAM chip temperature reaches steady-state quickly. This is because the time constant of the heat spreader (tens of seconds, as we show in the results section) is orders of magnitude larger than the DRAM chip die time constant (tens of milliseconds [Skadron et al. 2004]). We further simplify the model by assuming the power dissipation across ranks is uniform, as the memory controller applies interleaving by default to distribute the memory requests across the DIMMs and their ranks uniformly. The junction temperature of a given rank can be computed as

$$\frac{dT^D_j(t)}{dt} = -\frac{T^D_j(t)}{\tau^D_{ca}} + \frac{\gamma}{C^D_{ca}}\left(P^D + \frac{q^D}{\gamma}\right), \tag{5}$$

where $\gamma = (1 + \frac{R^D_j}{R^D_{ca}})$, $R^D_j = \frac{R^D_{chip}}{Nn_r}$, P^D represents the total operational power dissipated in the DIMM, τ^D_{ca} is the time constant of the heat spreader which equals $C^D_{ca}R^D_{ca}$.

3.3. Memory Consolidation: Energy Savings and Thermal Challenges

Modern DRAM modules are not energy proportional, since the energy consumed to process memory requests is only a fraction of the total module energy. This is because the DRAM consumes energy to maintain the state of the stored information and to provide adequate responsiveness. The DRAM energy can be saved by consolidating the active memory pages of the workload into a smaller set of active DIMMs and placing the rest in a self-refresh mode. However, we need to account for potential thermal

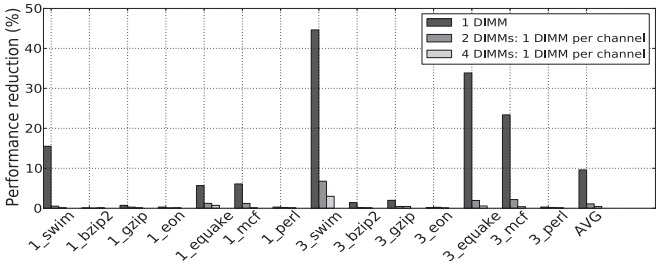


Fig. 4. Performance reduction with memory consolidation.

problems while consolidating, since clustering active pages in a smaller set of DIMMs elevates the power density.

We analyze the impact of consolidation on power per DIMM by running controlled experiments on our server (refer to Figure 3). For the default configuration, we use a total of eight DIMMs with two DIMMs per memory channel to maximize bandwidth. We emulate consolidation by using a configuration of four DIMMs, where each is connected to a separate memory channel to ensure maximum bandwidth. In these experiment we balance the workload across the dual CPU sockets. The default memory controller implements interleaving to maximize bandwidth by evenly distributing memory accesses across all memory channels. The results show that we can achieve 16W savings using consolidation. The memory power consumed in the default case is higher than half of the power consumed in the configuration of four DIMMs, because a fraction of the power is used to keep the DIMMs functional, which we call *baseline power*. Higher savings are achieved for memory-intensive jobs (i.e., *swim, equake, mcf*) compared to CPU-intensive, (i.e., *eon, gzip, perl, bzip2, gcc*) as expected. On the other hand, the consolidation caused the power density of the DIMMs to rise by up to 33%, which may lead to potential temperature problems and higher cooling energy costs.

We now study the impact of consolidation on performance where we use data collected from running experiments on our machine. For the default case, we assume a configuration of eight DIMMs, where every two DIMMs are connected to a separate memory channel. We examine the following configurations to emulate different degrees of consolidation: (a) single DIMM, (b) two DIMMs where each is connected to a memory channel that belongs to a separate memory controller, and (c) four DIMMs where each is connected to a separate memory channel. Figure 4 shows the effect of consolidation on performance compared to the default configuration. For the case of one- and two-DIMMs consolidation, the performance degradation is noticeable, since only a fraction of the bandwidth is utilized. However, when the memory bandwidth is fully utilized (i.e., case of four DIMMs), the resultant performance approaches the default case. The performance is slightly better in the default case compared to the that of four DIMMs due to the reduction in bank conflicts, as we are adding more banks. Nevertheless, this improvement is small since the number of pages (usually in the order of thousands or more) is much larger than the number of banks (order of tens), thus little improvement in temporal locality can be attained. The performance overhead of consolidation is acceptable when the memory bandwidth is fully utilized and the memory footprint of the workload fits in the active DIMMs.

4. COMBINED ENERGY, THERMAL, AND COOLING MANAGEMENT

In this section, we discuss the details of CoMETC. Figure 5 illustrates the framework of CoMETC, which consists of a formal multi-input multi-output (MIMO) controller, actuators (memory page scheduler, CPU socket scheduler, fan speed actuator), and

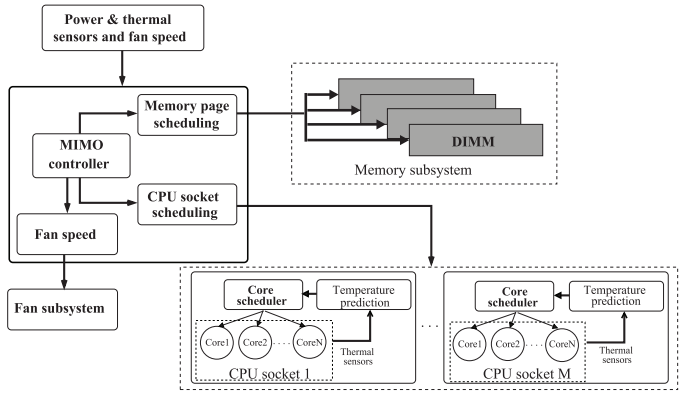


Fig. 5. Overview of the CoMETC framework.

sensors (temperature, power, and fan speed). Actuators translate the MIMO controller's decisions into proper actions on memory modules, CPU sockets, and fan. The controller takes feedback signals from the sensors to ensure convergence. We design our scheme in a unified fashion, since the temperature of memory is dependent upon the temperature of CPU. Two independent thermal management units for CPU and memory lead to inefficiencies. Formal control is used to guarantee efficiency and stability. The MIMO controller is implemented in the operating system layer. We design the MIMO controller using state-space control because it is robust and scalable [Franklin et al. 1990].

4.1. State-Space Control

We formulate a unified state-space model for memory and CPU subsystems. The vector of junction temperature of memory modules, $\mathbf{T}_j^D$, is defined as $[T_{j_1}^D(t), T_{j_2}^D(t), \ldots, T_{j_{n_D}}^D(t)]^T$, where n_D is the number of DIMMs. The vector of heat sources in the DIMMs, $\mathbf{U}^D$, is defined as $[P_1^D(t) + \frac{q_1^D(t)}{\gamma_1}, P_2^D(t) + \frac{q_2^D(t)}{\gamma_2}, \ldots, P_{n_D}^D(t) + \frac{q_{n_D}^D(t)}{\gamma_{n_D}}]^T$. Here, $P_i^D(t)$ and $q_i^D(t)$ correspond to the DIMM power dissipation and heat coupling, respectively. The value of γ_i is defined in Equation (5) as $1 + \frac{R_{j_i}^D}{R_{ca_i}^D}$. In case of no thermal coupling, we set $q_i^D(t)$ to zero, for $1 \leq i \leq n_D$.

Using Equation (5), we can express the thermal model for memory subsystem as

$$\frac{d\mathbf{T}_j^D(t)}{dt} = \mathbf{Y}^D\mathbf{T}_j^D(t) + \mathbf{Z}^\mathbf{D}\mathbf{U}^D(t), \tag{6}$$

where the temperature coefficient matrix, $\mathbf{Y}^D$, and input coefficient matrix, $\mathbf{Z}^D$, are defined as

$$\mathbf{Y}^D = \begin{bmatrix} \frac{-1}{\tau_{ca_1}^D} & & 0 \\ & \ddots & \\ 0 & & \frac{-1}{\tau_{ca_{n_D}}^D} \end{bmatrix}, \qquad \mathbf{Z}^D = \begin{bmatrix} \frac{\gamma_1}{C_{ca_1}} & & 0 \\ & \ddots & \\ 0 & & \frac{\gamma_{n_D}}{C_{ca_{n_D}}} \end{bmatrix}.$$

Similarly, we formulate the thermal model for a set of CPU sockets. The case to ambient temperature vector of CPUs, $\mathbf{T}_{ca}^C$, is defined as $[T_{ca_1}^C(t), T_{ca_2}^C(t), \ldots, T_{ca_{n_c}}^C(t)]^T$, where n_C is the number of CPU sockets. The vector for instantaneous power dissipation

in the CPU sockets, $\mathbf{P}^C$, is defined as $[P_1^C(t), P_2^C(t), \ldots, P_{n_C}^C(t)]^T$. Using Eq. (4), the CPU socket thermal model becomes

$$\frac{d\mathbf{T}_{ca}^C(t)}{dt} = \mathbf{Y}^C\mathbf{T}_{ca}^C(t) + \mathbf{Z}^C\mathbf{P}^C(t), \tag{7}$$

where the temperature coefficient matrix, $\mathbf{Y}^C$, and input coefficient matrix, $\mathbf{Z}^C$, are diagonal ($Y_{ii}^C = \frac{-1}{\tau_{ca_i}^C}$ and $Z_{ii}^C = \frac{1}{C_{ca_i}^C}$, for $1 \le i \le n_C$). The matrix $\mathbf{Y}^C$ is diagonal because the heat transfer between the CPUs is set to zero, as it is negligible (i.e., virtually no air flow passes across the CPUs, Figure 1(b)). The continuous linear systems given in Eqs. (6) and (7) are discretized using the transformations given in Franklin et al. [1990] as follows.

$$\mathbf{T}_j^D(k+1) = \mathbf{\Phi}^D\mathbf{T}_j^D(k) + \mathbf{\Gamma}^D\mathbf{U}^D(k), \tag{8}$$

$$\mathbf{T}_{ca}^C(k+1) = \mathbf{\Phi}^C\mathbf{T}_{ca}^C(k) + \mathbf{\Gamma}^C\mathbf{P}^C(k), \tag{9}$$

where the coefficients of this system are defined as follows

$$\mathbf{\Phi}^D = e^{\mathbf{Y}^D\Delta t}; \tag{10}$$

$$\mathbf{\Phi}^C = e^{\mathbf{Y}^C\Delta t}; \tag{11}$$

$$\mathbf{\Gamma}^D = \mathbf{Z}^D\int_0^{\Delta t} e^{\mathbf{Y}^D u}\,du; \tag{12}$$

$$\mathbf{\Gamma}^C = \mathbf{Z}^C\int_0^{\Delta t} e^{\mathbf{Y}^C u}\,du. \tag{13}$$

These coefficients are diagonal because they are functions of diagonal coefficients in the continuous system. Substituting $\mathbf{Y}^D$ and $\mathbf{Z}^D$ in Eqs. (10) and (12), we get

$$\mathbf{\Phi}^D = \begin{bmatrix} e^{-\frac{\Delta t}{\tau_{ca_1}^D}} & & 0 \\ & \ddots & \\ 0 & & e^{-\frac{\Delta t}{\tau_{ca_{n_D}}^D}} \end{bmatrix};$$

$$\mathbf{\Gamma}^D = \begin{bmatrix} \frac{\gamma_1}{C_{ca_1}}\left(\int_0^{\Delta t} e^{-\frac{u}{\tau_{ca_1}^D}}\,du\right) & & 0 \\ & \ddots & \\ 0 & & \frac{\gamma_{n_D}}{C_{ca_{n_D}}}\left(\int_0^{\Delta t} e^{-\frac{u}{\tau_{ca_{n_D}}^D}}\,du\right) \end{bmatrix}.$$

The integrals of $\mathbf{\Gamma}^D$ have analytical solutions as $\mathbf{\Gamma}_{ii}^D = (R_{ca_i}^D + R_{j_i}^D)(1 - e^{-\frac{\Delta t}{\tau_{ca_i}^D}})$, for $1 \le i \le n_D$. The matrices $\mathbf{\Gamma}^C$ and $\mathbf{\Phi}^C$ can be computed in a similar way.

Next we formulate a unified state-space model for the system using Eqs. (8) and (9), which yields

$$\begin{bmatrix} \mathbf{T}_{ca}^C(k+1) \\ \mathbf{T}_j^D(k+1) \end{bmatrix} = \begin{bmatrix} \mathbf{\Phi}^C & \mathbf{0} \\ \mathbf{\Phi}^{CD} & \mathbf{\Phi}^D \end{bmatrix}\begin{bmatrix} \mathbf{T}_{ca}^C(k) \\ \mathbf{T}_j^D(k) \end{bmatrix} + \begin{bmatrix} \mathbf{\Gamma}^C & \mathbf{0} \\ \mathbf{0} & \mathbf{\Gamma}^D \end{bmatrix}\begin{bmatrix} \mathbf{P}^C(k) \\ \mathbf{P}^D(k) \end{bmatrix}, \tag{14}$$

where $\mathbf{P}^D(k)$ corresponds to the vector of the DIMM's power consumption. The term $\mathbf{\Phi}^{CD}$ represents the thermal coupling coefficient from CPU sockets to memory modules. $\mathbf{\Phi}^{CD}_{ij} = \frac{\lambda_{ij}\Gamma^D_{ii}}{R^D_{ca_i}+R^D_{j_i}} \frac{R^C_{conv_j}}{R^C_{ca_j}}$, for $1 \leq i \leq n_D$ and $1 \leq j \leq n_C$, where λ_{ij} represents the thermal coupling factor between the DIMM i and the CPU j.

MIMO Controller. We use an MIMO controller as the primary management unit, which is implemented in the operating system layer. With a final target of minimizing the costs of cooling and memory energy, the MIMO controller takes all the important decisions for the (a) number of active DIMMs, (b) workload assignment or scheduling between CPU sockets, and (c) fan speed control that ensures convergence to the target temperatures of both CPU and memory subsystems with minimal performance overhead. The controller takes input using thermal and power sensors of CPU and memory, in addition to readings of fan speed. These sensors are usually available in state-of-the-art servers. As output, the MIMO controller provides a vector of desired power distributions for CPU and memory, along with a temperature vector that is used to determine fan speed. There are three actuators: CPU, memory, and fan. Actuators ensure that fan speed, power of CPU, and memory are set according to the controller's output.

The controller is designed based on the linear feedback control law [Franklin et al. 1990]. The control law is applied to the combined state-space model (Eq. (14)) which yields

$$\mathbf{P}(k) = -\mathbf{G}\mathbf{T}(k) + \mathbf{G_0}\mathbf{T}_r(k), \tag{15}$$

where $\mathbf{G}$ and $\mathbf{G_0}$ are the gain matrices with dimension $n \times n$ ($n = n_C + n_D$). The gain matrices are computed in a way that ensure efficiency and stability, as discussed in the subsequent paragraphs. The vector $\mathbf{P}(k)$ is the output of the controller which includes the power consumption values of CPUs and memory modules that need to be enforced by the actuators in the period between (k and $k+1$). Fan speed is set by the fan actuator based on the desired temperature distribution (temperature vector) specified by the controller. This temperature vector can be calculated by substituting Eq. (15) in Eq. (14), as

$$\mathbf{T}(k+1) = (\mathbf{\Phi} - \mathbf{\Gamma}\mathbf{G})\mathbf{T}(k) + \mathbf{\Gamma}\mathbf{G_0}\mathbf{T}_r(k), \tag{16}$$

where $\mathbf{\Phi} = \begin{bmatrix} \mathbf{\Phi}^C & \mathbf{0} \\ \mathbf{\Phi}^{CD} & \mathbf{\Phi}^D \end{bmatrix}$, $\mathbf{\Gamma} = \begin{bmatrix} \mathbf{\Gamma}^C & \mathbf{0} \\ \mathbf{0} & \mathbf{\Gamma}^D \end{bmatrix}$, vector $\mathbf{T}(k) = [\mathbf{T}^C(k), \mathbf{T}^D(k)]^T$, represents the thermal states of the system consisting of CPU, $\mathbf{T}^C(k)$, and memory temperatures, $\mathbf{T}^D(k)$. The target temperature vector $\mathbf{T}_r(k) = [\mathbf{T}^C_r(k), \mathbf{T}^D_r]^T$ consists of the target CPU, $\mathbf{T}^C_r(k)$, and memory temperatures, $\mathbf{T}^D_r$. The elements of $\mathbf{T}^D_r$ are the thermal emergency thresholds of the memory modules. The components of $\mathbf{T}^C_{r_i}(k)$ are calculated as follows

$$\mathbf{T}^C_{r_i}(k) = \mathbf{T}^C_{ca_i}(k) + \Delta T^C_{ca_i}(k) - \delta T^C_{th_i}(k), \tag{17}$$

where $\Delta T^C_{ca_i}(k) = \Delta R^C_{ca_i}(k) \frac{T^C_{ca_i}(k)}{R^C_{ca_i}(k)}$. $\Delta R^C_{ca_i}(k)$ is the change in the CPU case to ambient resistance, which relates to the difference between the current fan speed and the target speed. The difference between the junction temperature to the threshold is represented by $\delta T^C_{th_i}(k)$.

In general, the controller guarantees convergence to the desired target values, $\mathbf{T}_r(k)$, if the eigenvalues of the controlled feedback system are within the unit circle [Franklin et al. 1990]. One way to determine the feedback gain matrix, $\mathbf{G}$, is to use the desired eigenvalues as input and calculate $\mathbf{G}$ accordingly. To obtain the optimal gain matrix, we can use the linear quadratic regulator (LQR) optimization [Franklin et al. 1990]. This

method calculates the gain matrix in a way that minimizes the following cost function:

$$J = \sum_{k=0}^{\infty} [\mathbf{T}^T(k)\mathbf{Q}\mathbf{T}(k) + \mathbf{u}^T(k)\mathbf{R}\mathbf{u}(k)], \tag{18}$$

where $\mathbf{u} = -\mathbf{G}\mathbf{T}(k)$. $\mathbf{Q}$ and $\mathbf{R}$ are symmetric weight matrices of size $n \times n$, and they are specified by the designer. They are selected based on the importance of the states and the energy of the control outputs, respectively. The LQR computations are done offline. It takes around one second to compute a gain matrix for different fan speeds. The gain matrix is stored as an array and accessed at runtime. The input gain matrix $\mathbf{G_0}$ is calculated using the standard reference input method described in Franklin et al. [1990]. The controller interval is on the order of several seconds, since the thermal time constant of the DIMMs and the CPUS is on the order of tens of seconds.

Design Overhead. Extracting the thermal model and designing the controller is a one-time cost. Once the design is developed, it can be reused across the servers in the deployment; the reuse makes this overhead acceptable. We rely on the capability of our MIMO controller to handle any small deviations in the characteristics between the machines [Franklin et al. 1990].

4.2. Actuators

At each controlling tick, k, the MIMO controller determines the set of operating power values for the CPUs, memory modules, and fan speed for the next interval (k and $k+1$). The output of the controller is communicated to the actuators to execute the controller requests. Each actuator operates independently, as the MIMO controller already considers the thermal dependencies between the components.

Controller Convergence in the Face of Errors. The actuator's goal is to minimize the differences between the calculated power values provided by the controller and the current measured ones. When the action of the actuator has some occasional errors, then the controller can still ensure convergence. To validate our assumptions we assume a state space model that is similar to the CPU and memory. We use a CPU model as an illustrative example.

$$T_{ca}^C(k+1) = \Phi T_{ca}^C(k) + \Gamma P^C(k), \tag{19}$$

where Φ and Γ are coefficients and P^C is the input power of a given CPU. To control this system, we use the state-space control that we have in Eq. (15). Let's assume a reference point to be 0 to simplify the analysis. The feedback control in this case can be computed as $P^C(k) = -GT_{ca}^C(k) + e(k)$. G is the gain and $e(k)$ is the actuator average error. The accumulated error in $T_{ca}^C(k)$ at $k = n$ equals to $\sum_{i=0}^{n} \Gamma e(i)\upsilon^{n-i}$, where υ is the eigenvalue of the controller. Since the eigenvalue of the controller is less than 1, the accumulated error converges to 0, which ensures convergence. The details of the actuators implementation are given next.

CPU Actuator. The controller's input to the CPU actuator is the vector $\Delta\mathbf{P^C}$ of the desired change in power values in each of the CPUs for the interval between k and $k+1$. This vector is calculated by subtracting the CPUs power requested by the controller from the CPU's average power measured in the interval between $k-1$ and k. The details of the CPU scheduler are given in Algorithm 1. The algorithm starts by estimating the power consumed by the individual threads in each CPU via modeling their core and last-level cache power, similar to what is proposed in Ayoub et al. [2011]. Subsequently, our algorithm traverses the workload and spreads the threads from the hot CPU, starting with cooler threads to have a finer-grain control of the total power in

ALGORITHM 1: Socket-Level Scheduling

Calculate ΔP^C and the set of P^C_{thr} for each CPU. Set Q as an empty queue;
for *i in the set of hot CPUs* **do**
  for *j in the set of threads in* CPU_i **do**
    dest $\Leftarrow$ index of the coolest CPU;
    if $(P^C_{thr_j} \leq \Delta P^C_i)$ *and* $(P^C_{thr_j} \leq |\Delta P^C_{dest}|)$;
    then
      if CPU_{dest} *has idle core* **then**
        Calculate cooling savings of migrating thread j to CPU_{dest};
      else
        Calculate cooling savings of swapping thread j with the coolest thread in CPU_{dest};
      end
      if *cooling savings* $> S_{min}$ **then**
        Enqueue this migration event in Q;
        Update ΔP^C and threads assignment;
      end
    end
  end
end
Execute all migration events in Q

each socket and to reduce the chance of errors. It moves from a hot CPU_i a number of threads with a total power that is less than or equal to ΔP^C_i. A cool CPU_{dest} receives threads with a total power that is less than or equal to $|\Delta P^C_{dest}|$. Before each migration, we evaluate the cooling energy savings to prevent ineffective scheduling decisions, similar to Ayoub et al. [2011]. The cooling energy savings estimator calculates the resultant temperature after the migration using our thermal model. Subsequently, we translate the difference in maximum temperature into change in fan speed using Equation (23), which we use in the fan actuator. Individual scheduling events are allowed only when the predicted cooling savings are higher than a given threshold, S_{min}.

Memory Actuator. At the beginning of each interval, the controller provides a vector, $\mathbf{P}^D$, of desired power dissipation per each DIMM. As a result, a DIMM may need to keep, increase, or decrease its power dissipation accordingly. A page migration mechanism is used as a proxy to control the active power of the DIMMs as required. When the temperature is high and no active DIMMs can accept additional pages to reduce the power density of the hot DIMMs, then we need to choose between activating a new DIMM or spinning up the fan to cool down the DIMMs. We decide between these two options based on the temperature reduction each choice can deliver under the same energy budget. In the following sections, we study these scenarios.

(1) *Increasing Fan Speed Versus Activating a New DIMM*. Figure 6 shows that doubling the number of DIMMs does not reduce the power per DIMM to half. This is due to the baseline power, P_{base}, component that is in the range of 3.5 W for memory bound applications measured on 4GB DDR2 DIMMs (refer to Section 3.3). This means that increasing fan speed may be a better option if it results in a lower temperature at a power consumption of P_{base}. Increasing fan speed reduces the memory temperature by lowering the effect of thermal coupling and self heating of the DIMMs, as it reduces the convective resistance of the CPU and memory. The

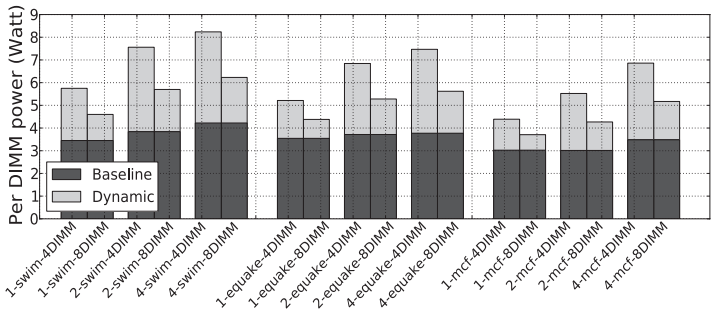


Fig. 6. Power breakdown of memory DIMMs.

temperature reduction for the increase in fan power by P_{base} is calculated as

$$\Delta T_{fan} = \Delta F \left(\lambda P^C_{act} \frac{dR^C_{conv}(F)}{dF} + P^D_{act} \frac{dR^D_{conv}(F)}{dF} \right), \tag{20}$$

where ΔF is the increase in fan speed of the hottest memory zone. P^C_{act} and P^D_{act} represent the actual power consumption of CPU and the power of hottest DIMM (located in the same CPU zone), respectively. When the actuator decides to activate a new DIMM, it migrates pages from other DIMMs to the newly activated DIMM, progressively starting from the application that has the highest memory access rate. Using our thermal model, the temperature reduction for adding one more DIMM can be computed as

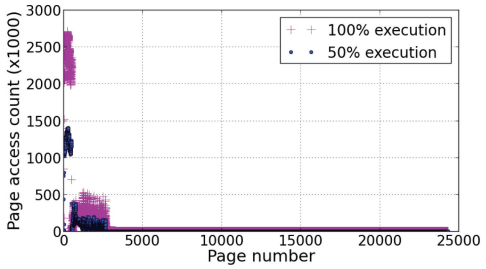
$$\Delta T_{mem} = \frac{\left(T^D_{j_{max}} - P_{base} R^D_{ca} - \lambda T^C_{ha}\right)}{n_D}, \tag{21}$$

where n_D is the number of DIMMs after expansion and λ is the thermal coupling factor between CPU and memory. The $T^D_{j_{max}}$ is the maximum temperature of the DIMMs that are located in the zone of the given CPU socket. The controller chooses to activate a new DIMM only if the temperature reduction, $|\Delta T_{mem}|$, is higher than $|\Delta T_{fan}|$ when a new DIMM is activated, as follows.

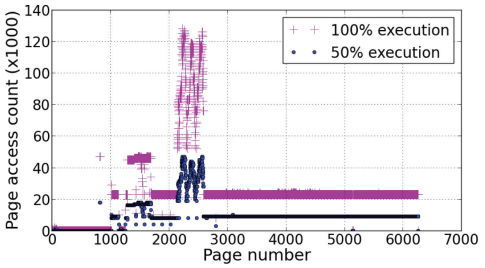
$$\begin{aligned} &\textbf{if}(|\Delta T_{fan}| < |\Delta T_{mem}|) \Rightarrow \textit{activate a new DIMM}, \\ &\textbf{else} \Rightarrow \textit{increase fan speed}. \end{aligned} \tag{22}$$

We optimize for page migration by exploiting the skew in the distribution of page activity. Figures 7(a), 7(b), and 7(c) show the page access patterns for memory and CPU bound applications, for each application, we give the access distribution for half and full of the simulated interval. We use a microarchitectural simulator M5 [Binkert et al. 2006] to generate these statistics and simulate for a representative period of 15 billion instructions of each benchmark. From these results, it can be seen that the number of active memory pages (*hot pages*) is a small fraction of the total pages in the application. When the actuator decides to migrate pages from hot DIMMs, we migrate the active pages to mitigate temperature with a minimal number of page migrations.

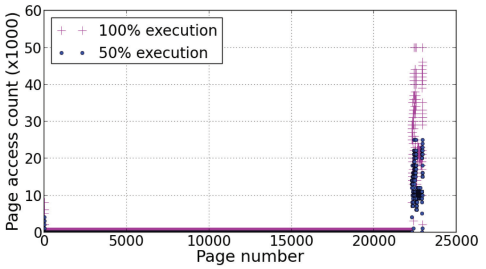
(2) *Controlling DIMM's Power by Page Migration.* Power vector $\mathbf{P}^D(k)$ specifies power distribution of each DIMM over the next period. The power of a memory module i needs to be reduced in the next interval when $\mathbf{P}^D_i(k) < \mathbf{P}^D_{meas_i}(k-1)$, where $\mathbf{P}^D_{meas_i}(k-1)$ is the measured power of DIMM$_i$ in the previous interval. This power reduction in DIMM$_i$ is achieved by migrating portion of its pages to other DIMMs.



(a) mcf (memory bound).



(b) equake (memory bound).



(c) bzip2 (CPU bound).

Fig. 7. Access pattern of memory pages.

This migration is performed by using the memory actuator. In contrast, the power dissipation of a memory module, say DIMM$_i$, may also need to be increased when $\mathbf{P^D}_i(k) > \mathbf{P}^D_{meas_i}(k-1)$. Hence it requires migrating more pages to DIMM$_i$ from other memory modules to match DIMM$_i$'s power with the desired value by the controller.

When all active memory modules can accept more pages, the controller starts migrating from the most recently activated module so that it can be placed in a low power mode when no active pages are left. When there are some hot DIMMs, the controller tries to maximize performance by migrating pages to a memory module only if it has lower memory access rate/power consumption than the average value. In this way, all the accesses are uniformly distributed among the active memory modules. To minimize hot spots, the controller balances the number of active DIMMs per CPU zone (CPU and its associated downstream DIMMs). For example, the activation order of DIMMs shown in Figure 1(b) should be $A_i, B_i, C_i, D_i, A_{i+1}, B_{i+1}$, etc. The page migration continues until the access rate of the newly activated DIMM becomes equal to that of already active DIMMs. The overhead of migrating pages is acceptable since the temperature time constant is over six orders of magnitude larger than the page migration time.

Fan Actuator. Fan speed is updated periodically based on the requested temperatures from the controller, $\mathbf{T}(k+1)$. Controlling the fan is used as a complementary measure to the CPU and memory optimizations to minimize the remaining difference between the current measured temperature and the controller's requested temperature vector.

Let's assume $\Delta\mathbf{T}^D > \mathbf{0}$ and $\Delta\mathbf{T}^C > \mathbf{0}$ are vectors of the temperature difference between the current and target value of the controller for CPU and memory modules, respectively. The actuator estimates the new fan speed, F_{new}, based on the current fan speed, F_{cur}, and highest requested change in fan speed due to memory, ΔF^D, and due

to CPU, ΔF^C, as follows.

$$\Delta F^D = max\left(\frac{\Delta T_i^D}{\lambda P_j^C \frac{dR_{conv_j}^C(F)}{dF} + P_i^D \frac{dR_{conv_i}^D(F)}{dF}}\right); \tag{23}$$

$$\Delta F^C = \frac{\Delta T_j^C}{P_j^C \frac{dR_{conv_j}^C(F)}{dF}}; \tag{24}$$

$$F_{new} = F_{cur} + max(\Delta F^D, \Delta F^C), \tag{25}$$

where i represents the ith DIMM that is in the zone of processor j. The interval of the fan actuator can be set to be equal to or less than the controller interval. Setting the fan actuator interval to a smaller value helps provide more detailed control over temperature. The desired temperature at the subintervals is computed based on the slop of $\mathbf{T}(k)$ and $\mathbf{T}(k+1)$.

5. EVALUATION

5.1. Methodology

We evaluate our approach using actual power traces (memory and CPU) collected from our instrumented server. We feed these traces to an extended version of HotSpot simulator [Skadron et al. 2004] to estimate the temperature of the CPU and memory subsystems for a given cooling rate. We extended HotSpot with our unified thermal/cooling model (refer to Figure 2). We estimate the CPU core, L2, and baseline power using the method described in Ayoub et al. [2011]. The core and L2 power traces are used to estimate the core and L2 temperatures, respectively, using HotSpot simulator and Xeon processor layout. We include the baseline power of the CPU in the temperature simulations.

We use the memory organization given in our Intel server (see Figure 1(b)). To save energy, the memory controller put the DIMMs with dormant pages in a self-refresh mode, while the rest of the modules remain active. The values of power consumption during self-refresh and transition penalty are given in Table I. We generate the page access statistics of the applications using the M5 microarchitectural simulator [Binkert et al. 2006].

The simulation parameters of the fan and thermal package of both CPU and memory DIMMs are listed in Table I. The cooling parameters for both CPU and memory are calibrated using transient measurements similar to those used to generate Figures 9(a) and 9(b).

The default *fan control algorithm* is modeled as a closed loop PI (proportional and integral) controller, which is commonly used in modern systems. The fan controller sets the fan speed in proportion to the difference to thermal threshold and the accumulated temperature errors. We set the fan to trigger five degrees below the thermal threshold of the chip (refer to Table I) to allow enough time to respond to thermal emergencies. We use simulation instead of running our algorithms in a real system due to implementation constraints. The built-in fan control algorithm spins all fans at a single speed that is required to mitigate the temperature of the hottest CPU or DIMM, which leads to over-provisioning. This algorithm is implemented in a separate microcontroller that we don't have access to. As a results, not all the advantages of our algorithm can be manifested using the built-in fan control algorithm.

We set the sampling interval for the MIMO controller to four seconds, this is a reasonable assumption, since the temperature time constant of the CPU heat sink and DIMM

Table I. Characteristics of CPU, Memory, and Cooling

CPU	CPU	Xeon E5440
	TDP	80W
	CPU frequency	2.8GHz
	Heat spreader thickness	1.5mm
	Case to ambient thermal resistance in K/W	$R^C_{ca} = 0.141 + \frac{1.23}{V^{0.923}}$, V: air flow in CFM
	Heat sink time constant at max air flow	25 seconds
	Temperature threshold	90°C [Patterson 2008]
DIMM	DIMM size	4GB
	Max DRAM power/DIMM	10W
	Case to ambient thermal resistance in K/W	$R^D_{ca} = 0.75 + \frac{45}{V^{0.9}}$ V: air flow in CFM
	Per chip thermal resistance in K/W	$R^D_{chip} = 4.0$[2]
	Heat spreader time constant at max air flow	70 seconds
	Temperature, threshold	85°C [Lin et al. 2007, 2009]
	Self refresh power per DIMM	0.15W
	Transition between active and self-refresh modes	11us [Hai et al. 2005]
	Thermal coupling factor with CPU	0.65
Fan	Fan power per socket	29.4 W
	Max air flow rate per socket	53.4 CFM
	Fan steps	32
	Fan sampling interval	1 second
	Idle fan speed	10% of max speed

Table II. SPEC Benchmarks Characteristics

Benchmark	IPC	Power per DIMM (W)	Characteristics
swim	0.55	4.65	Memory bound
equake	0.51	4.38	Memory bound
mcf	0.18	3.71	Memory bound
perl	2.18	1.21	CPU bound
bzip2	1.36	1.18	CPU bound
gcc	1.23	1.17	CPU bound
eon	1.33	0.79	CPU bound
gzip	1.16	0.92	CPU bound

heat spreader are on the order of tens of seconds. It should be noted that our technique is not restricted to this interval and other intervals around this value can be used. The sampling interval of the fan control is set to one second to allow for a fine-grain control over temperature to ensure reliability. We also set the cooling savings threshold to a conservative value of 10%. For core-level thermal management, we use our previously proposed OS-level proactive thermal management [Ayoub and Rosing 2009].

For workload, we use a set of benchmarks that have a wide range of CPU and memory activity to emulate real-life applications (see Table II). Each benchmark is executed till completion, then repeated until a total execution time of 600 seconds is reached after a warm-up period of 200 seconds. Table III gives the list of the workload combinations that we use in this study. In these experiments, we run a representative workload that has a mix of CPU and memory bound applications.

[2]http:www.micron.com/products/dram/.

Table III. Workload Combinations for Multitier Algorithm

Workload	Socket A	Socket B	Workload	Socket A	Socket B
WL1	*equake + 2gzip*	*3bzip2*	WL9	*mcf + 2gcc*	*perl + bzip2 + eon*
WL2	*2bzip2 + 2eon*	*mcf + gcc + 2gzip*	WL10	*3gzip*	*2perl + eon*
WL3	*equake + 2bzip2*	*2gzip + gcc*	WL11	*2equake + gcc*	*2perl + equake*
WL4	*perl + eon + bzip2*	*2equake + gcc*	WL12	*mcf + 2gcc*	*2gcc + 2perl*
WL5	*2gcc + perl*	*swim + 2gcc*	WL13	*gcc + 2perl*	*equake + 2gcc*
WL6	*mcf*	*mcf*	WL14	*2mcf + gcc*	*2perl + mcf*
WL7	*gcc + 2gzip*	*2bzip2 + perl*	WL15	*2swim + gcc*	*2perl + swim*
WL8	*perl + bzip2 + 2eon*	*2mcf + 2gcc*			

The CoMETC algorithm employes an MIMO control to manage workload scheduling across CPU sockets, distribution of memory access between DIMMs, and controlling fan speed. It also implements proactive thermal management at the core level to mitigate temperature within sockets. We set the minimum number of active DIMMs to four to utilize full memory bandwidth (refer to Figure 1(b)). We evaluate CoMETC against the following policies.

Thermal Management of Core and Memory with PI Fan Control, CM+PI. This is the default policy which implements state-of-the-art thermal management techniques for CPU and memory. It employs proactive thermal management at the core level to mitigate the temperature within individual sockets [Ayoub and Rosing 2009]. Memory thermal problems are managed separately via throttling the memory access when the temperature reaches the emergency threshold [Lin et al. 2007]. No memory clustering is applied in this policy. The fan speed is controlled separately using the default PI controller.

Joint Energy, Thermal, and Cooling Management JETC [Ayoub et al. 2012]. This policy implements workload scheduling between sockets, memory clustering with a tradeoff optimization between fan speed and memory clustering. This trade-off optimization is similar in concept to what we use in CoMETC. However, it uses heuristics instead of formal MIMO control for scheduling jobs between CPU sockets and managing memory clustering. The fan speed is controlled separately using the default PI control. The details of these heuristics can be found in Ayoub et al. [2012], which we omit due to space limitation. For core-level thermal management, it employs the same proactive thermal technique that is used in CoMETC.

No Fan-Memory Optimization, NFMO. This policy inherits the same properties of CoMETC, except we disable the optimization that trades off between activating a new DIMM and increasing fan speed. This policy is selected to study the impact of this trade-off on savings.

No Memory Management, NMM. This policy is similar to CoMETC, except we preclude the memory clustering optimization. The purpose of this policy is to isolate the effect of memory clustering on savings.

5.2. Results

5.2.1. Evaluation of Thermal and Cooling Model. First, we evalutate the thermal and cooling model.

Thermal Coupling. We start by addressing the thermal coupling between CPU and memory using measurements collected from our machine. To do so, we collect the temperature of the CPU heat sink via external thermal sensors and the maximum junction temperature of the memory modules using the Intelligent Platform Management

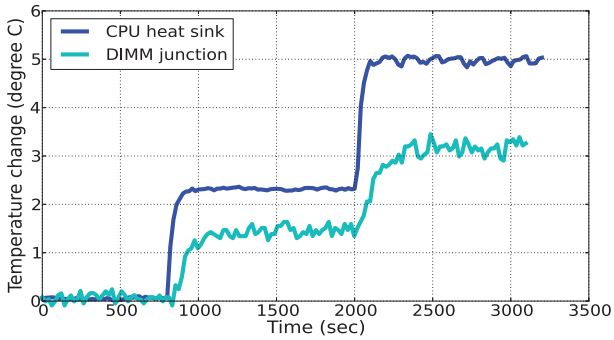
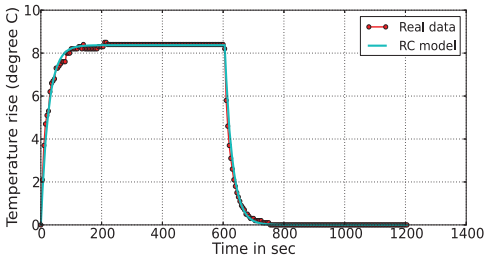
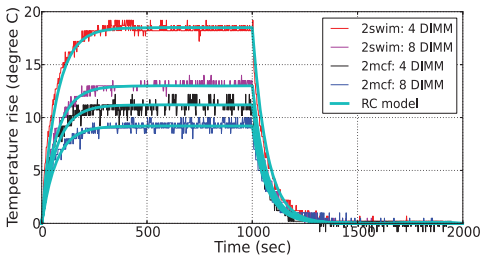


Fig. 8. Thermal coupling between CPU and memory.



(a) CPU's T_{ha}^{C} transient temperature.



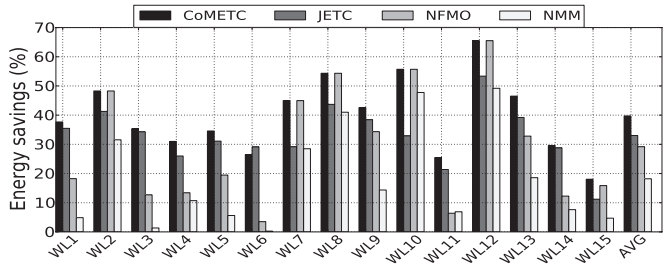
(b) DIMM's transient temperature.

Fig. 9. Transient temperature for CPU heat sink and DIMMs (referenced to idle temperature).

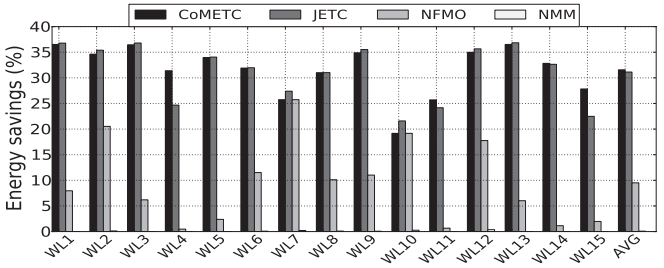
Interface (IPMI), both at the same time. In this experiment, we use four DIMMs, each connected to a single channel, where every two DIMMs are placed behind one CPU socket. We run *swim* (memory bound) alone for 300 seconds on a single socket as a warmup. After this warm-up period, we continue running *swim* for another 800 seconds then add *perl* (CPU bound) and launch an additional *perl* at time 2,000 seconds, all to the same socket. The rationale behind running *swim* is to keep the memory power at the same level during the experiment since it has almost a flat power. During this experiment, we restart any finished jobs. Figure 8 shows the temperatures of the memory and the CPU heat sink referenced to their respective values at the end of the warm-up period. The results clearly show that a rise in the heat sink temperature causes a rise in the memory temperature due to the extra heat that is transferred from the CPU to memory.

CPU Thermal Model. Next, we validate the CPU thermal model using an experimental setup similar to that of Ayoub et al. [2011]. In this experiment, we insert an external thermal sensor to the heat sink surface which measures the temperature from the heat sink to the ambient, T_{ha}^{C}. We run four instances of *perl* for 600 seconds followed by an additional 600 seconds of idleness for cooling down. The fan speed is set to maximum using the boost mode option to keep the time constant of the heat sink constant. Figure 9(a) shows measured and modeled values of T_{ha}^{C}. It can be seen from the results that there is a strong match between the measurements and the model.

Memory Thermal Model. Now we validate our memory model using measurements collected from the machine. We execute memory-intensive jobs, *swim* and *mcf*, and collect the junction temperature of the DIMMs using IPMI. We validate the model for two cases, four DIMMs and eight DIMMs configurations to assure that our model is



(a) Local ambient temperature is 45^{o}C.



(b) Local ambient temperature is 35^{o}C.

Fig. 10. Total energy savings (memory + cooling) relative to the default policy in a system with eight DIMMs.

not limited to a certain configuration. The four DIMMs and eight DIMMs are placed in the memory slots following the order (A_1, B_1, C_1, D_1) and (A_1, A_2, B_1, B_2, C_1, C_2, D_1, D_2), respectively (refer to Figure 1(b)). The fan speed is set to maximum using the boost mode option to keep the time constant of the heat spreader of the memory modules constant. Figure 9(b) shows the transient temperature of the measured and modeled values for four and eight DIMMs configurations. The figure shows clearly that the transient temperature of the memory modules is dominated by the thermal dynamics of the heat spreader as it changes slowly (time constant $\sim$70 seconds). The results clearly show a decent match between the actual and ideal model with an average error of 0.27°C relative to the real measurements.

5.2.2. Evaluation of the CoMETC Algorithm. Next, we evaluate the CoMETC algorithm and compare it against other policies. In the following, we evaluate energy savings, fan balancing, page-migration rate, stability, and overhead.

Energy Savings. First, we address the energy savings of CoMETC compared to other policies. The savings results include the energy of the memory and cooling subsystems. Figures 10(a) and 10(b) show the total energy savings for eight-DIMM configuration with local server ambient temperatures of 45°C and 35°C, respectively. CoMETC achieves energy savings reaching an average of 39.7% and 31.6% relative to the default policy (CM + PI) for 45°C and 35°C, respectively. The results in Figure 10(a) clearly show that CoMETC outperforms all other policies. For instance, the savings in the cases of WL7 and WL10 are related to imbalance in the thermal distribution between the two sockets, which raises cooling energy costs. The CoMETC algorithm outperforms the default policy quite well, since it has the capability to balance the temperature between the CPU sockets while the default is restricted to local thermal optimizations within the individual sockets only. The policy NFMO performs equally well to CoMETC, since it can balance the temperature between sockets and uses MIMO control that is

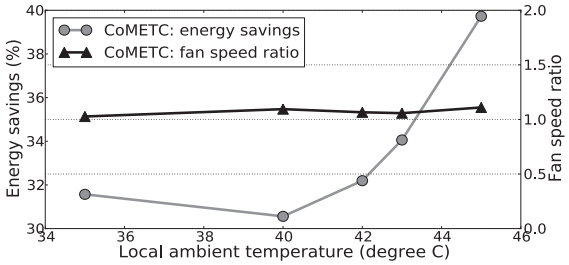


Fig. 11. Ambient temperature sensitivity of energy savings and fan speed ratio with eight DIMMs.

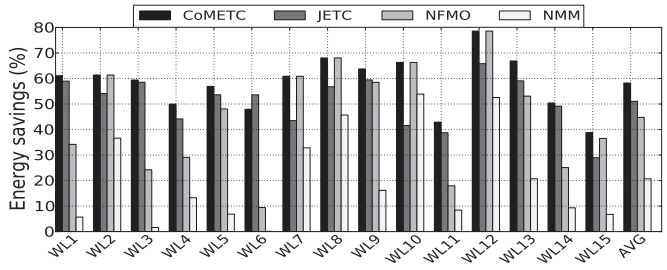
similar to CoMETC. However, JETC delivers noticeably lower savings, albeit it has the capability to balance the heat between sockets. JETC laas behind these policies since it does not integrate fan control with workload scheduling and uses heuristics which can not guarantee optimality. The policy NMM also as behind CoMETC since it does not implement memory clustering to save on memory energy. The other important scenario is when there are savings opportunities from CPU temperature imbalance and memory clustering. Examples of this can be seen in the cases of WL4 and WL5. In these cases, CoMETC is superior to all other policies, since it is the only one that can capture these classes of savings efficiently.

Next, we present the combined energy savings of memory and cooling at a local ambient temperature of 35°C, shown in Figure 10(b). The savings in this case are dominated by clustering the memory requests to a smaller set of DIMMs. Evidence of this can be seen from the savings of the NMM policy, which is close to zero. The CoMETC policy performs well since it is able to capture the savings opportunities from memory clustering. These results also illustrate the advantages of the optimization that trades off between activating a new DIMM versus speeding up the fan. This can be indicated from the low savings of NFMO, since it does not perform this optimization. The JETC policy performed close to CoMETC, because performing the memory clustering optimization at low temperature does not require a highly optimized control, as thermal problems are infrequent.

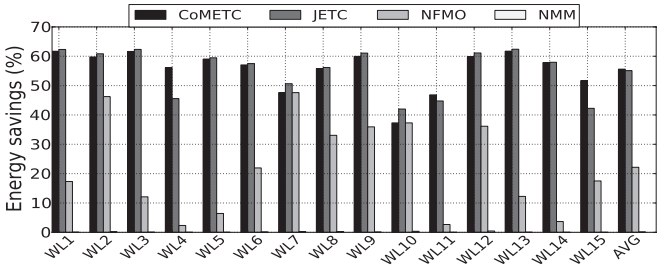
In Figure 11, we show the energy savings of CoMETC over the default policy as a function of local ambient temperature. The savings between 35°C and 40°C come primarily from clustering the memory accesses to a subset of the memory modules. When the local ambient temperature increases, the savings become higher due to balancing fan speed, as both memory and CPU experience more thermal issues. This indicates that CoMETC provides better savings at higher local ambient temperatures.

In Figures 12(a) and 12(b), we study the benefits of increasing the number of DIMMs to 16. The CoMETC algorithm is able to achieve higher savings compared to the default policy, reaching an average of 58.2% and 55.6% for local ambient of 45°C and 35°C, respectively. The savings of CoMETC increase when using 16 DIMMs, as compared to the case of eight DIMMs since more DIMMs can transition to low-power modes. The CoMETC algorithm outperforms the JETC policy for the case of the 16 DIMMs as well. The results also show that the relative increase in savings with respect to the case of eight DIMMs is higher in the case of 35°C compared to that of 45°C for CoMETC. This is because the savings in the case of 35°C are dominated by the memory subsystem, while the memory contributes to only a fraction of savings in the case of 45°C. In summary, the results clearly show that performing thermal management in a holistic manner leads to large energy savings.

We now study the breakdown in energy savings. Figure 13 depicts the energy savings breakdown between cooling and memory. Each bar in the graph is tagged as policy



(a) Local ambient temperature is 45^{o}C.



(b) Local ambient temperature is 35^{o}C.

Fig. 12. Total energy savings (memory + cooling) relative to the default policy in a system with 16 DIMMs.

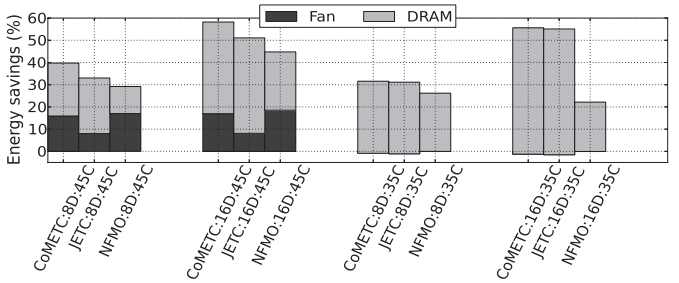
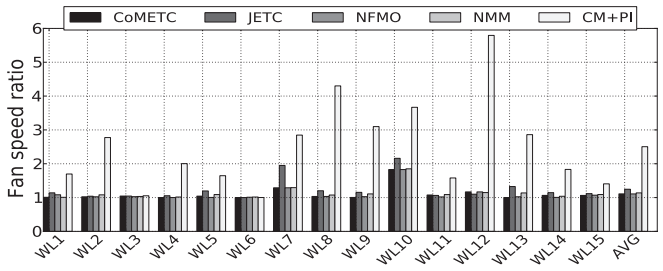
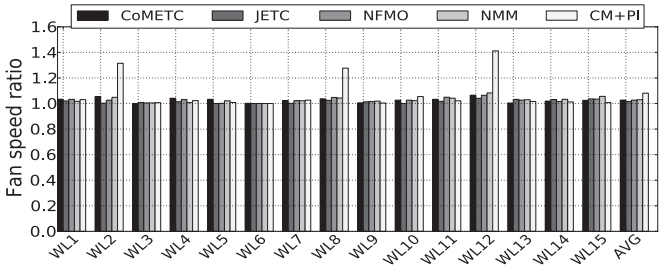


Fig. 13. Energy savings breakdown.

name, number of DIMMs, and local ambient temperature (e.g., CoMETC:8D:45C, which corresponds to the CoMETC policy, eight DIMMs and 45°C ambient). The CoMETC outperforms JETC for 45°C ambient in cooling savings, since CoMETC optimizes for cooling better as the fan control speed is unified with the CPU scheduling and memory clustering within the MIMO control, whereas the cooling in JETC is controlled separately via a PID control. On the other hand, the CoMETC policy achieved noticeably higher memory savings but slightly worse in cooling savings compared to NFMO for 45°C ambient. This is because NFMO does not account for the trade-off between memory and fan energy. The savings in the case of 35°C ambient come from memory clustering only, as there is no potential savings from cooling due to the low temperature in the system. The memory savings of CoMETC and JETC is comparable, while the cooling savings is slightly on the negative side, at 35°C ambient. The cooling savings are marginally on the negative side, since these polices increase the fan speed slightly to prevent activating new DIMMs while keeping the net savings positive. The savings from NFMO is noticeably lower since it does not take the advantage from this optimization.



(a) Local ambient temperature 45^oC.



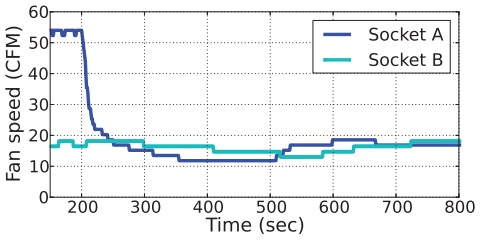
(b) Local ambient temperature is 35^oC.

Fig. 14. Fan speed ratio in a system with eight DIMMs.

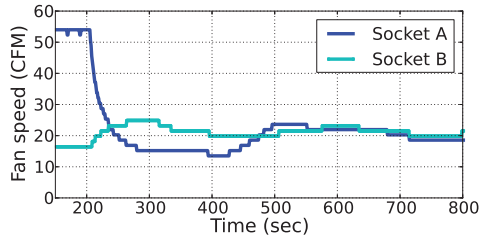
Fan Balancing. Next we evaluate CoMETC for balancing the speed of fans in the system. The optimal ratio of fan speed between two sockets is 1 (we call it the *fan speed ratio*). Figure 14(a) shows the fan speed ratio for a system with eight DIMMs and 45°C local ambient temperature. In this case, the CoMETC policy is able to reduce the standard deviation from the optimal fan speed ratio by 84% compared to the CM + PI policy. The CM + PI policy performs poorly since it does not balance the temperature between CPU sockets. The other polices (JETC, NFMO, NMM) perform close to CoMETC as they implement socket-level thermal balancing. Although the fan ratio for JETC is close to that for CoMETC, the actual savings of the CoMETC is higher, for as it minimizes the absolute value of fan speeds, which translates into higher savings, as we show in Figure 13. Figure 14(b) shows the fan speed ratio results in a system with eight DIMMS and 35°C ambient temperature. In this scenario, the CoMETC policy achieved a reduction in standard deviation from the optimal target of 86% compared to CM + PI. Figure 11 shows sensitivity analysis of average fan speed ratio as a function of local ambient temperature for CoMETC. The results clearly show that CoMETC is able to keep the fan ratio close to 1.0 for the entire temperature range between 35°C and 45°C.

In Figures 15(a), 15(b), and 15(c), we study the benefits of CoMETC on balancing fan speeds for the illustrative set of workloads WL13, WL11, and WL15, respectively. In these experiments, we run the workload for 200 seconds (with no core, socket, or memory scheduling), then we initiate CoMETC and allow it to run for 600 seconds. We select illustrative workloads which represent a mix of CPU and memory-intensive jobs to evaluate CoMETC's effect on CPU, memory, and thermal coupling. It is clear from the results that CoMETC is able to converge and balance the fan speed in all cases.

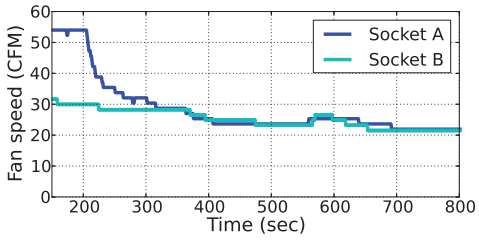
Page Migration. Next, we quantify the page-migration rate which is another important factor in our design. Figure 16(a) depicts the rate of page migration with eight-DIMM configuration and 45°C ambient temperature. The data show that the



(a) Workload W13.

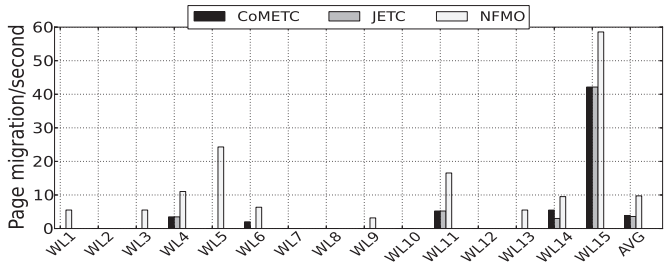


(b) Workload W11.

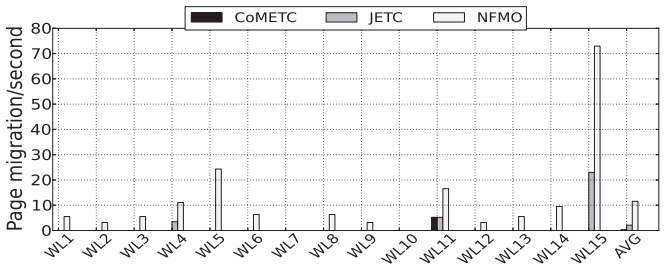


(c) Workload W15.

Fig. 15. Fan speed transient response with CoMETC, eight DIMM, and 45°C.



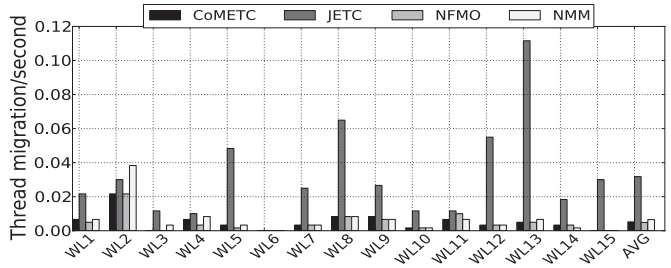
(a) Local ambient temperature is 45°C.



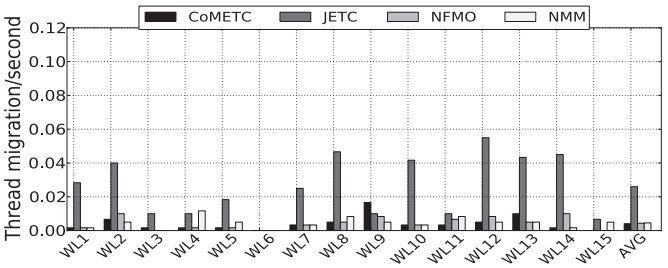
(b) Local ambient temperature is 35°C.

Fig. 16. Page-migration rate for a system with eight DIMMs.

average rate of page migrations of CoMETC is smaller than five pages/second. This overhead is negligible in practice, since migrating a page takes only a few microseconds. The NFMO policy has the highest page-migration rate since it does not optimize between increasing fan speed versus activating a new DIMM; hence, it may activate more DIMMs compared to CoMETC, which leads to higher migrations. The rate of page migration of the JETC policy is comparable to that of CoMETC, since JETC optimizes



(a) Local ambient temperature is 45^oC.



(b) Local ambient temperature is 35^oC.

Fig. 17. Thread migration rate between CPU sockets for a system with eight DIMMs.

for the trade-off between fan speed and memory clustering. The case of WL15 has the highest migration rate, since the workload mix has three instances of *swim* (*swim* has a wide set of hot pages). The page-migration rate for the case of eight DIMMs and 35°C ambient temperature is shown in Figure 16(b). In this case, the migration rate for CoMETC approaches zero, which causes virtually no overhead.

Thread Migrations between CPU Sockets. The other important metric we analyze is the rate of thread migrations across CPU sockets. Figure 17(a) depicts the rate of thread migrations between the two CPU sockets with eight-DIMM memory configuration and 45°C ambient temperature. The results show that the average rate of thread migrations for CoMETC is about 0.005 thread/second, which is insignificant. The rate of thread migrations of the CoMETC policy is noticeably smaller compared to JETC since CoMETC implements a more stable control; an illustrative example can be seen in the case of workload WL13. The scenario of workload WL6 triggers no thread migrations in any of the policies, since both CPU sockets have balanced power dissipation (i.e., both run similar workload). Figure 17(b) depicts the rate of thread migrations between the two CPU sockets at a lower ambient temperature of 35°C. The average rate of thread migrations in this case is smaller compared to the case of 45°C due to the reduction in the occurrence of high temperature events.

Performance Overhead. Table IV shows the breakdown of the average performance overhead for the policies studied in this work with eight-DIMM and 16-DIMM memory organizations and 45°C and 35°C ambient temperatures. The data in this table include the overhead of CPU scheduling (i.e., thread migrations across and within CPU sockets), page migrations, and execution throttling. We account for the OS and microarchitectural performance costs in the computation of these overheads. The migration of a page from one DIMM to another involves a number of events which we account for—the primary events are allocating a free page in the destination DIMM, copying the content of the page from the source DIMM to the destination one, updating the

Table IV. Performance Overhead (%)

Setup	Overhead source	CoMETC	JETC	NFMO	NMM	DTM-CM+PI
45°C	Schedule across CPU sockets	0.0001	0.0004	0.0001	0.0001	—
ambient	Schedule within CPU sockets	0.0728	0.0548	0.0806	0.0886	0.0673
	Page migration	0.0002	0.0002	0.0006	—	—
8 DIMM	Throttling	0.0019	0.0161	0.0021	0.0036	0.0022
35°C	Schedule across CPU sockets	0.0001	0.0003	0.0001	0.0001	—
ambient	Schedule within CPU sockets	0.0157	0.0149	0.0192	0.0265	0.0200
	Page migration	0.0000	0.0001	0.0007	—	—
8 DIMM	Throttling	0.0000	0.0001	0.0000	0.0323	0.0000
45°C	Schedule across CPU sockets	0.0001	0.0004	0.0001	0.0001	—
ambient	Schedule within CPU sockets	0.0729	0.0557	0.0837	0.0994	0.0745
	Page migration	0.0002	0.0002	0.0008	—	—
16 DIMM	Throttling	0.0019	0.0124	0.0022	0.0077	0.0043
35°C	Schedule across CPU sockets	0.0001	0.0003	0.0001	0.0000	—
ambient	Schedule within CPU sockets	0.0156	0.0143	0.0212	0.0336	0.0231
	Page migration	0.0000	0.0001	0.0014	—	—
16 DIMM	Throttling	0.0000	0.0001	0.0000	0.0000	0.0000

page table in the OS, invalidating the cache and TLB entries corresponding to the page in the source DIMM, and releasing the memory page in the source DIMM. Regarding thread migration between cores within the same CPU socket, we account for the following events: copy the state of the source thread to the destination core, write back the dirty lines in the L1 cache to the last-level cache, and cold start execution. The migration of a thread across two sockets incurs a higher overhead, since we need to write back the dirty lines corresponding to the source thread in the L1 and last-level cache to the memory, and account for a higher cold start overhead.

The average overhead of CoMETC at 45°C is 0.225%, which is insignificant (including memory clustering overhead of 0.15%). The performance overhead of CoMETC is small, as it optimizes for CPU and memory scheduling and also exploits the slowness in temperature change that enables lower scheduling frequency. The migration and throttling performance overheads are lower for 35°C ambient compared to 45°C ambient due to infrequent thermal problems in the case of 35°C ambient. The results in Table IV show a negligible overhead of page migration for CoMETC in most cases due to its associated small page-migration rate (e.g., five pages/sec for the case of eight DIMMs and 45°C ambient, refer to Figure 16(a)). The page-migrations overhead of NFMO is higher than CoMETC, as NFMO does not optimize between activating a new DIMM versus increasing fan speed, which may result in a higher page-migrations rate. Regarding the CPU scheduling, the CoMETC policy does a good job in keeping this overhead small, as shown in Table IV. The CoMETC overhead of thread migrations between CPU sockets is insignificant in all cases, since the rate of thread migrations is kept very small (refer to Figure 17). This small overhead is contributed to the stability of the CoMETC controller and the long time constant of the heat sink. The results also show that the overhead of thread migrations within the CPU sockets for CoMETC is small, below 0.08% in all cases. The CoMETC and other policies incur a relatively comparable overhead of thread migrations within the CPU sockets, since all employ the same policy, as we discussed earlier. In summary, the overall scheduling overhead of CoMETC is kept in a similar range of the default policy while providing substantial energy savings (memory + cooling) of 58% over the default policy.

6. CONCLUSION

Prior research handled temperature and computing energy problems separately and also did not consider cooling, resulting in suboptimal solutions. In this work, we develop

CoMETC, a unified solution that integrates energy, thermal, and cooling management decisions to maximize energy savings. As part of CoMETC, we have proposed a unified thermal and cooling model for CPU and memory subsystems that is suitable for online management. Our solution is based on a formal MIMO control to ensure efficiency and stability. CoMETC reduces the operational energy of the memory by clustering memory requests to a subset of memory modules while considering thermal and cooling metrics. It also removes hot spots between and within the sockets and reduces the effects of thermal coupling to minimize cooling costs. CoMETC also controls fan speed at the same time to maximize savings and ensure stability. Our experimental results show that CoMETC can deliver an average cooling and memory energy savings of 58% compared to state-of-the-art techniques at a negligible performance overhead.

REFERENCES

AJAMI, A., BANERJEE, K., AND PEDRAM, M. 2005. Modeling and analysis of nonuniform substrate temperature effects on global interconnects. *IEEE Trans. Comput. Aid. Des. Integr. Circ. Syst. 24*, 6, 849–861.

AYOUB, R., INDUKURI, K. R., AND ROSING, T. S. 2011. Temperature aware dynamic workload scheduling in multisocket cpu servers. *IEEE Trans. Comput. Aid. Des. Integr. Circ. Syst. 30,* 9, 1359–1372.

AYOUB, R., NATH, R., AND ROSING, T. 2012. JETC: Joint energy thermal and cooling management for memory and cpu subsystems in servers. In *Proceedings of the IEEE 18th International Symposium on High Performance Computer Architecture (HPCA)*. 1–12.

AYOUB, R. AND ROSING, T. S. 2009. Predict and act: Dynamic thermal management for multi-core processors. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 99–104.

BARROSO, L. AND HOLZLE, U. 2007. The case for energy-proportional computing. *IEEE Computer 40,* 12, 33–37.

BARROSO, L. AND HOLZLE, U. 2009. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, Morgan and Claypool Publishers.

BINKERT, N., DRESLINSKI, R., L. HSU, K. L., SAIDI, A., AND REINHARDT, S. 2006. The M5 simulator: Modeling networked systems. *IEEE Micro 26*, 4, 52–60.

CHIUEH, H., LUH, L., DRAPER, J., AND CHOMA, J. 2000. A novel fully integrated fan controller for advanced computer systems. In *Proceedings of the Southwest Symposium on Mixed-Signal Design (SSMSD)*. 191–194.

CHOI, J., CHER, C., FRANKE, H., WEGER, A., AND BOSE, P. 2007. Thermal-aware task scheduling at the system software level. In *Proceedings of the 12th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED).* 213–218.

COSKUN, A., ROSING, T., AND GROSS, K. 2008. Proactive temperature management in mpsocs. In *Proceedings of the 13th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED).*

DONALD, J. AND MARTONOSI, M. 2006. Techniques for multicore thermal management: Classification and new exploration. In *Proceedings of the 33rd International Symposium on Computer Architecture (ISCA)*. 78–88.

FAN, X., ELLIS, C., AND LEBECK, A. 2001. Memory controller policies for DRAM power management. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 129–134.

FRANKLIN, G., POWEL, J., AND WORKMAN, M. 1990. *Digital Control of Dynamic Systems*. Addison-Wesley.

HAI, H., KANG, S., CHARLES, L., AND TOM, K. 2005. Improving energy efficiency by making dram less randomly accessed. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 393–398.

HEATH, T., CENTENO, A. P., GEORGE, P., RAMOS, L., JALURIA, Y., AND BIANCHINI, R. 2006. Mercury and freon: Temperature emulation and management for server systems. In *Proceedings of the 12th International Conference on Architectural Support Programming Languages and Operating Systems (ASPLOS)*. 106–116.

HEO, S., BARR, K., AND ASANOVIC, K. 2003. Reducing power density through activity migration. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*. 217–222.

LIN, C.-H., YANG, C.-L., AND KING, K.-J. 2009. PPT: Joint performance/power/thermal management of DRAM memory for multi-core systems. In *Proceedings of the 14th ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*. 93–98.

LIN, J., ZHENG, H., ZHU, Z., DAVID, H., AND ZHANG, Z. 2007. Thermal modeling and management of dram memory systems. In *Proceedings of the 34th International Symposium on Computer Architecture (ISCA)*. 312–322.

LIN, J., ZHENG, H., ZHU, Z., GORBATOV, E., DAVID, H., AND ZHANG, Z. 2008. Software thermal management of dram memory for multicore systems. In *Proceedings of the ACM SIGMETRICS Conference on Mesurement and Modeling of Computer Systems (SIGMETRICS)*. 337–348.

PATTERSON, M. 2008. The effect of data center temperature on energy efficiency. In *Proceedings of the 11th Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITHERM)*. 1167–1174.

PEDRAM, M. AND NAZARIAN, S. 2006. Thermal modeling, analysis, and management in VLSI circuits: Principles and methods. *Proc. IEEE 94*, 8, 1487–1501.

SCHMIDT, R., CRUZ, E., AND IYENGAR, K. 2005. Challenges of data center thermal management. *IBM J. Res. Devel. 49,* 4/5, 709–723.

SHIN, D., KIM, J., CHOI, J., CHUNG, S., AND CHUNG, E. 2009. Energy-optimal dynamic thermal management for green computing. In *Proceedings of the International Symposium on Computer-Aided Design (ICCAD)*.

SKADRON, K., STAN, M., HUANG, W., VELUSAMY, S., SANKARANARAYANAN, K., AND TARJAN, D. 2003. Temperature-aware microarchitecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA)*. 2–13.

SKADRON, K., STAN, M., SANKARANARAYANAN, K., HUANG, W., VELUSAMY, S., AND TARJAN, D. 2004. Temperature-aware microarchitecture: Modeling and implementation. *ACM Trans. Architect. Code Optim. 1*, 1, 94–125.

SONG, L., LEUNG, B., NECKAR, A., MEMIK, S., MEMIK, G., AND HARDAVELLAS, N. 2011. Hardware/software techniques for DRAM thermal management. In *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 515–525.

TANG, Q., GUPTA, S., AND VARSAMOPOULOS, G. 2007. Thermal-aware task scheduling for data centers through minimizing heat recirculation. In *Proceedings of the IEEE International Symposium on Cluster Computing (ICCC)*. 129–138.

TOLIA, N., WANG, Z., MARWAH, M., BASH, C., RANGANATHAN, P., AND ZHU, X. 2009. Unified thermal and power management in server enclosures. In *Proceedings of the ASME InterPACK Conference Collacated with ASME Summer Heat Transfer Conference and the ASME 3rd International Symposium on Energy Sustainibility (IPACK)*. 1–12.

WANG, Z., BASH, C., TOLIA, N., MARWAH, M., ZHU, X., AND RANGANATHAN, P. 2009. Optimal fan speed control for thermal management of servers. In *Proceedings of the ASME/Pacific Rim Technical Conference and Exhibition on Packaging and Integration Electronic and Photoic Systems, MEMS, and NEMS (InterPACK'09)*. 1–10.

WEI, H., MALCOLM, A.-W., JOHN, C., ELMOOTAZBELLAH, E., HENDRIK, H., TOM, K., CHARLES, L., JIAN, L., KARTHICK, R., AND JUAN, R. 2011. TAPO: Thermal-aware power optimization techniques for servers and data. In *Proceedings of the International Green Computing Conference and Workshops (IGCC)*. 1–8.

YEO, I., LIU, C., AND KIM, E. 2008. Predictive dynamic thermal management for multicore systems. In *Proceedings of the Design Automation Conference (DAC)*. 734–739.

Received August 2012; revised April 2013; accepted June 2013