JETC: Joint Energy Thermal and Cooling Management for Memory and CPU Subsystems in Servers

Raid Ayoub Rajib Nath Tajana Rosing
University of California, San Diego
La Jolla, CA 92093-0404

Abstract

In this work we propose a joint energy, thermal and cooling management technique (JETC) that significantly reduces per server cooling and memory energy costs. Our analysis shows that decoupling the optimization of cooling energy of CPU & memory and the optimization of memory energy leads to suboptimal solutions due to thermal dependencies between CPU and memory and non-linearity in cooling energy. This motivates us to develop a holistic solution that integrates the energy, thermal and cooling management to maximize energy savings with negligible performance hit. JETC considers thermal and power states of CPU & memory, thermal coupling between them and fan speed to arrive at energy efficient decisions. It has CPU and memory actuators to implement its decisions. The memory actuator reduces the energy of memory by performing cooling aware clustering of memory pages to a subset of memory modules. The CPU actuator saves cooling energy by reducing the hot spots between and within the CPU sockets and minimizing the effects of thermal coupling. Our experimental results show that employing JETC results in 50.7% average energy reduction in cooling and memory subsystems with less than 0.3% performance overhead.

1. Introduction

Technology scaling coupled with the high demand for computation leads to a wide use of server systems with multiple CPU sockets and larger amounts of memory resulting in higher power densities [1, 2]. High power dissipation increases the operational costs of machines. It also causes thermal hot spots that have substantial effect on reliability, performance and leakage power [25, 7]. Dissipating the excess heat is a big challenge as it requires a complex and energy hungry cooling subsystems.

Traditionally, the CPU is known to be the primary source of system power consumption. Over the years the designers have developed techniques to improve the energy efficiency of the CPU subsystem which accounts for approximately 50% of the total energy budget. Less attention is given to energy optimization in rest of the system components which leads to a poor energy proportionality of the entire system [11]. Memory subsystem is the other major power hungry component in the server systems as it consumes up to 35% of total system energy and has poor energy proportionality [10, 11]. Capacity and bandwidth of the memory subsystem are typically designed to handle the worst case scenarios. Applications vary significantly in terms of their access rates to the memory. It is common that only a fraction of memory pages is active while the rest are dormant. One solution is to activate a subset of the memory modules that can serve the applications needs to save energy [18]. However, this approach increases the power density of the active memory modules which can cause thermal problems in the memory.

In [21, 22], the authors proposed a solution to mitigate thermal emergencies in the memory system by adjusting memory throughput to keep the memory temperature in the safe zone. However, this solution does not improve energy proportionality in the memory subsystem since it does not consider minimizing the number of active DIMMs to just what is needed. To manage thermal emergencies within a single CPU socket, a number of core level dynamic thermal management (DTM) techniques have been proposed [15, 30]. The benefits of these techniques are limited to managing the temperature within a single socket. The important issue is that none of the existing techniques consider the dynamics of the cooling subsystem and its energy costs.

Modern servers incorporate a fan subsystem to reduce their temperature. The power consumed by a fan is cubically related to the air flow rate which makes it energy hungry [24]. The fan system in high-end servers consumes as much as 80 Watts in 1U rack servers [24] and 240 Watts or more in 2U rack servers [1]. The cooling subsystem becomes inefficient when it operates far from the optimal point due to poor thermal distribution. Workload allocation policies need to account for cooling characteristics to minimize cooling energy by creating a better thermal distribution. Due to cost and area constraints, a common set of

fans is normally used to cool both the CPUs and memory [2]. For such scenarios, the inlet temperature of the downstream (components at the end of air flow path) become a function of the temperature of the upstream (components at the entrance of air flow) in addition to the primary inlet temperature of the server. Energy optimization techniques must consider temperature constraints and cooling costs for making intelligent decisions.

In this work we present JETC, a joint energy, thermal and cooling management algorithm for individual servers which targets the improvements in server power usage efficiency (SPUE) by dynamically reducing the cooling and memory energy costs. Providing an integrated solution is necessary due to the thermal dependencies between the CPU and memory when both share same cooling resources. JETC maximizes the energy efficiency in the server by controlling the number of active memory modules to just what is needed to provide sufficient storage capacity and bandwidth while minimizing the cooling costs. JETC also schedules the workload between the CPU sockets to create a more balanced thermal distribution between them not only to minimize the thermal coupling effect but also to mitigate their thermal hot spots as well. Finally, we show that using JETC results in 50.7% average energy reduction of memory and cooling subsystems with less than 0.3% performance overhead.

2. Related work

In recent years, energy and thermal management in memory and CPU has become a vibrant research area. The research in this domain can be classified broadly into three main categories as follows:

Power and thermal management of memory subsystem: Recent research has sought a number of techniques to optimize the energy consumption in memory subsystems. The work in [17] lowers DRAM power by changing memory mode to low power state when there are no accesses to the memory. This approach is beneficial only when there are frequent idle periods. Research in [18] manages memory power by clustering heavily accessed pages to a subset of the memory modules and putting the rest in a self-refresh mode. However, this consolidation is likely to generate thermal problems due to clustering. Both of these techniques do not consider any thermal issues.

To handle thermal problems in memory subsystem, a few techniques have been suggested [21, 20]. The work in [21] mitigates overheating in memory system by adjusting memory throughput to stay below the emergency level. The primary drawback of this approach is the associated performance overhead. Research in [20] manages memory overheating by grouping threads in the system in such a way that each group is mapped to a subset of memory models assuming that all the threads in a group can be active simultaneously. Only one group of memory models is active at any point in time while the rest stay inactive to cool. The authors assume that the bandwidth of each thread is known in advance through static profiling which is normally not the case in dynamic systems. Additionally, this approach is restricted to the cases when number of threads is a multiple of the number of cores. The work in [23] proposes thermal a management technique for the memory subsystem at the microarchitectural level. This technique does not optimize for number of active DIMMs to reduce energy. However, none of these techniques account for cooling energy which can be a significant portion of the system energy.

Thermal management of CPU subsystem: In recent years, a number of core level thermal management techniques have been suggested. The research in [13] proposes two reactive DTM techniques. The first technique is based on dynamic voltage-frequency scaling while the other uses pipeline throttling to remove the excess heat. Recently, DTM has become an integral part of actual processor designs. For example, Intel Xeon processor employs clock gating and dynamic voltage-scaling to manage thermal emergencies. The research in [14, 19, 16] proposes activity migration to manage the excess temperature by moving computations across replicated units. The drawbacks with all of these approaches are performance overhead and poor thermal distribution since they are reactive in nature. To overcome the problems with reactive techniques, a class of proactive thermal management techniques have been suggested that try to avoid the overheating while the processor is still running below the temperature threshold. The authors in [15] proposed a temperature prediction model that is based on the serial autocorrelation in the temperature time series data. Although their model is fairly accurate, it requires a run time training phase that could impact performance. In addition, all these techniques are limited to a single CPU socket and do not model cooling dynamics which may have a significant effect on the temperature profile.

Cooling management: Recently, a few cooling algorithms have been suggested to optimize the fan controller [29, 26]. The research in [29] suggests an optimal fan speed mechanism for the blade servers that determines the optimal speed through solving a convex optimization. The work in [26] designs a fan control mechanism that also considers the leakage power. A class of techniques have been suggested to improve cooling efficiency at the data center level [28]. These techniques suggest the use of workload scheduling to mitigate the air circulation problem in data centers. However, they are not effective at the socket level since the air flow is highly contained within servers. In [24, 3] methodologies are proposed to model the convective thermal resistance of the heat sink as a function of the air flow rate, which we use here.

In this work we make the following main contributions:

- We present JETC, a novel integrated energy, thermal and cooling management for memory and CPU subsystem to deliver high energy savings.
- We have developed a detailed memory thermal model that considers the cooling dynamics and thermal coupling with other hot spots in the system. This is the first comprehensive model that integrates the cooling dynamics and thermal coupling between CPU and memory subsystems.
- We present a thorough evaluation of the our technique that achieves 50.7% average energy savings in cooling and memory subsystems at performance overhead of less than 0.3%.

3. Combined thermal and cooling model for CPU and memory

Figure 1(a) depicts the photo of our 45nm Intel Quad Core dual socket Xeon E5440 server where the CPU is placed close to the fan while the memory is downstream [2]. Having a shared cooling resource creates thermal dependencies between the upstream and downstream components. This server supports two off-chip memory controllers where each is connected to memory by two memory channels, each channel is connected to four dual in-line memory modules (DIMMs) as shown in Figure 1(b), each is a 4GB DDR2. We measure power of individual DIMMs with data acquisition system by adding extenders that support current sensors in the supply lines of each DIMM. The specifications of the CPU, memory and cooling subsystems are given in Table 1. Benchmarks from the SPEC2000 suite have been used as workloads.

3.1. System thermal model

Cooling systems use fans to generate air flow to reduce the temperature of hot components. In our dual socket system, the generated air flow passes through the heat sink of the CPU first and eventually reaches the memory DIMMs. This means that the memory and CPU are thermally connected as the CPU heats up the common air flow before it reaches the memory subsystem. As a result, the temperature of the memory is a strong function of the hottest CPU and the inlet temperature.

Both cooling and thermals are modeled based on the known duality between electricity and temperature [27]. Cooling is commonly modeled through variable convective resistance, where its value is a function of the air flow rate [24]. The value of the convective resistance R_{conv} can be computed as:

$$R_{conv} \propto \frac{1}{AV^{\alpha}} \quad (1)$$

where A is the heat sink effective area, 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. To estimate the power dissipation of fan we use the results from [24] to relate the fan speed, F, with the air flow rate as: $V \propto F$. The cooling power costs for changing the air flow rate from V_1 to V_2 can be computed as [24, 26]:

$$\frac{P_{V2}}{P_{V1}} = (\frac{V_2}{V_1})^3 \quad (2)$$

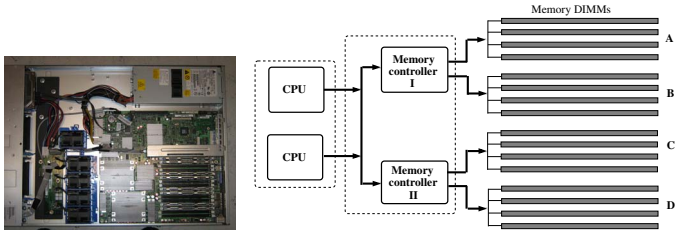
where P_{V1} and P_{V2} represent the fan's power dissipation at V_1 and V_2 respectively. Figure 2 show the unified thermal/cooling model of both CPU and memory. We combine CPU and memory thermal model using dependent thermal coupling heat sources to model the extra heat that is generated by the upstream CPU. Definitions of CPU and memory thermal models are discussed in sections 3.2 & 3.3 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 thermal resistance of the case to ambient of the memory DIMMs, R^D_{ca}, as follows:

$$q^D \propto \frac{T^C_{ha}}{R^D_{ca}} \quad (3)$$

Experimental verification of thermal coupling: We measure the temperature of the CPU heat sink which is located on the upstream of air flow using a thermal sensor. At the same time we measure the maximum junction temperature of the memory using Intelligent Platform Management Interface (IPMI) to observe the correlation between heat sink temperature and that of memory. We also measure the power of the DIMMs to account for temperature changes due to power variations. For this experiments we use 4 DIMMs, each connected to a single channel, where every two DIMMs are located after one CPU socket. We run a single threaded memory intensive workload from SPEC2000 suite, *swim*, on a single socket followed by progressively adding a CPU intensive workload, *perl*, to the same socket. The rationale behind running *swim* is to keep the memory power at the same level during the experiment. After a warm up period of 300 seconds we continue run swim alone for 800 seconds and execute one instance of *perl* right after this period and we add additional instance of *perl* at time 2000 seconds. During this experiment, we restart any finished jobs. Figure 3 shows the temperatures of memory and the CPU heat sink that are referenced to their 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 transfered to memory.

3.2. CPU thermal and cooling model

Unlike memory DIMMs where each DIMM contains multiple chips, the CPU socket usually has a single chip. However, the power distribution within the CPU socket is not uniform (e.g. some cores may be active while others are



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

Figure 1. Intel dual socket Xeon server

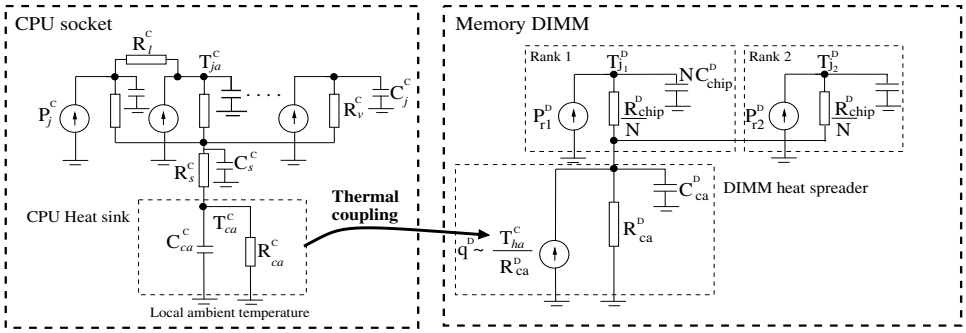


Figure 2. Combined thermal model

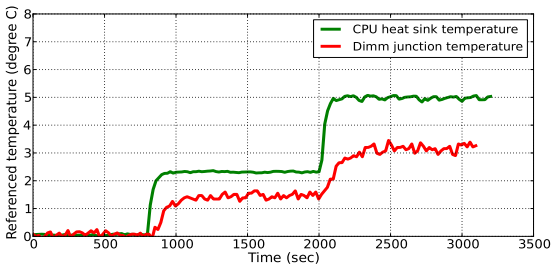


Figure 3. Thermal coupling

idle). The left part of Figure 2 shows the thermal model of the CPU chip with a thermal package [8, 27]. The thermal model of the CPU includes the die and the heat spreader. R_v^C is the die component's vertical thermal resistance including the thermal interface resistance. The lateral heat flow between die components is modeled using a lateral thermal resistance R_l^C, but its effect is negligible [19]. C_j^C is the thermal capacitance of the components in the die. The power dissipated by individual components is modeled as a heat source, P_j^C. R_s^C and C_s^C refer to the thermal resistance and capacitance of the heat spreader respectively.

The heat flow from the CPU case to local ambient (ambient temperature is measured inside the server enclosure) is modeled as a combination of conduction and convection heat transfers, [27]. The thermal conductive resistance of the heat sink is R_{hs}^C. The convective part is modeled by a convective resistance, R_{conv}^C, connected in series with R_{hs}^C where their sum represents the case to ambient resistance, R_{ca}^C. R_{ca}^C is connected in parallel with the thermal capacitance of the heat sink, C_{ca}^C, to form a simple RC circuit. The changes in R_{conv}^C with air flow is calculated using (1).

3.3. Memory thermal and cooling model

Typical DRAM subsystem is organized as an array of multiple DIMMs (see Figure 1(b)), where each DIMM is composed of ranks, usually 2, and each rank contains multiple memory chips (e.g. 8 chips per rank). Each DRAM chip in a rank stores data that belongs to a predetermined segment of bits in the data word. As a result, the rank power is distributed almost equally across the DRAM chips inside the rank. To enhance the heat transfer between the DRAM chips and local ambient the DIMMs have a heat spreader.

The right part of Figure 2 shows the thermal model of a DIMM that uses a heat spreader. We use superposition theory to simplify the RC network of the DRAM chips of each rank to a single RC node. In this Figure, the heat source P_{chip}^D equals to the power dissipated in each DRAM chip, R_{chip}^D is the vertical thermal resistance of each chip, C_{chip}^D is the thermal capacitance of each chip and, T_j^D is the junction temperature of a DRAM chip. The number of DRAM chips in each rank is assumed to be N.

The heat transfer between the heat spreader and local ambient is modeled as an RC node as shown in Figure 2. The thermal capacitance between the heat spreader and ambient is represented by C_{ca}^D. The value of R_s^D represents thermal resistance of the heat spreader. The heat transfer between the heat spreader and the interior ambient of the machine is modeled by a convective resistor, R_{conv}^D. The component R_{ca}^D corresponds to the case to ambient thermal resistance ($R_{ca}^D = R_s^D + R_{conv}^D$). The time constant of heat spreader is in the range of tens of seconds [21] while the time constant for the DRAM chip die is in the order of tens of milliseconds [27]. This means that the transient

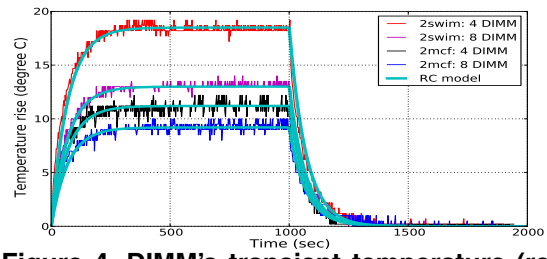


Figure 4. DIMM's transient temperature (referenced to idle temperature)

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. We exploit the big differences in time constants to reduce the thermal model complexity. We can simplify the model further by assuming that the power across ranks is similar since the memory controller uses interleaving to uniformly distribute the activities across the DIMMs as well as their ranks, n_r. The junction temperature of a given rank can be modeled as:

$$\frac{dT_j^D(t)}{dt} = -\frac{T_j^D(t)}{\tau_{ca}^D} + \frac{\gamma}{C_{ca}^D}(P^D + \frac{q^D}{\gamma}) \quad (4)$$

where $\gamma = (1 + \frac{R_j^D}{R_{ca}^D})$ and $R_j^D = \frac{R_{chip}^D}{Nn_r}$. The term P^D represents the total power dissipated in the DIMM and τ_{ca}^D is the time constant of the heat spreader component which equals to $C_{ca}^D R_{ca}^D$.

Experimental verification of DIMM thermal model We run memory intensive benchmarks, *swim* and *mcf* from SPEC2K, and collected the DIMMs junction temperature traces using IPMI. Figure 4 shows the temperature traces of 4 DIMMs and 8 DIMMs configurations to show that our model is not limited to a specific setup. The 4 DIMMs and 8 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 as shown in Figure 1(b). The fan speed is set to maximum using boost mode option to keep the time constant of the heat spreader of the memory modules constant. The figure shows clearly that the DIMM temperature dynamics is dominated by the heat spreader temperature since it changes slowly, with a time constant of 70 seconds. We plotted the transient temperature of the RC model and compared it with the actual temperature readings to show that the case to ambient temperature changes can be modeled using a single RC node as shown in Figure 4. The results show a strong match between the actual and ideal model with average error of 0.27^{o}C relative to the real measurements.

3.4. Sources of energy savings in memory

The DRAM energy consumed to do the actual job of reading or writing data is only a fraction of the total DRAM energy. This is because DIMMs consume energy to keep state of the stored information and to provide sufficient responsiveness to the incoming requests [18]. We can save energy by consolidating the active memory pages of the workload into a smaller set of DIMMs and place the rest in a self-refresh mode [18].

Figure 5 shows the impact of consolidation on power per DIMM (The number in front of the benchmark refers to the number of instances we run). In this experiment we balance the workload across the two sockets. In the first configuration, we use four DIMMs where each is connected to a separate memory channel to maximize bandwidth. For the second case, we use eight DIMMs with two DIMMs per channel. We can achieve 16Watts savings with this consolidation, higher savings can be delivered when the system has more DIMMs. The power is not decreased to half in the case of eight DIMMs which indicates that a fraction of the memory power is used to keep the DIMMs functional, which we call *baseline power*. The savings are higher for memory intensive workloads (*swim, equake, mcf*) compared to CPU intensive ones (*eon, gzip, perl, bzip2, gcc*) as expected. However, this consolidation increases the power density by up to 33% which can elevate the temperature problems.

Next we address the impact of consolidation on performance. We run experiments on our machine with different DIMMs organizations as follows: single DIMM, two DIMMs and four DIMMs (each DIMM is connected to a separate memory channel) and eight DIMMs (every two DIMMs are connected to a separate memory channel). Figure 6 shows the effect of DIMM consolidation on performance compared to a system with eight DIMMs. For the case of using one and two DIMMs the performance degradation is noticeable since only a fraction of the bandwidth is utilized. However, when we fully utilize the memory bandwidth (as in the case of four DIMMs) the resultant performance is close to the baseline case. This slight improvement in the case of eight DIMMs compared to four DIMMs is related to the reduction in bank conflicts as we are adding more banks. Nevertheless, this improvement is small since the number of pages (usually order of thousands or more) is much larger than the number of banks (order of tens). Hence, little improvement in temporal locality can be attained. This indicates that the performance overhead due to consolidation is in the acceptable range assuming the mem-

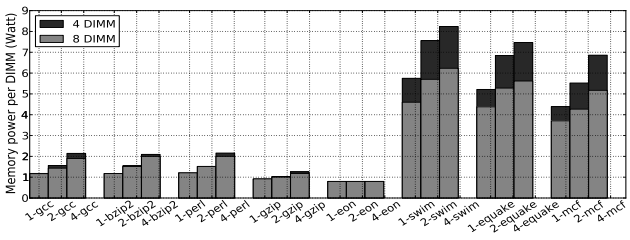
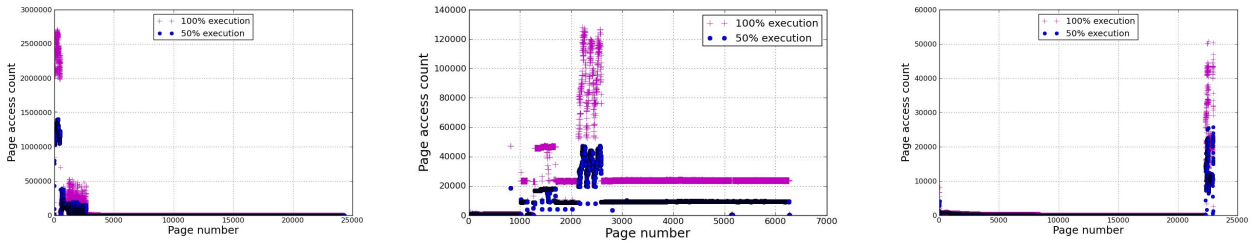


Figure 5. Memory power per DIMM



(a) mcf (memory bound) (b) equake (memory bound) (c) bzip2 (CPU bound)

Figure 7. Memory page access pattern

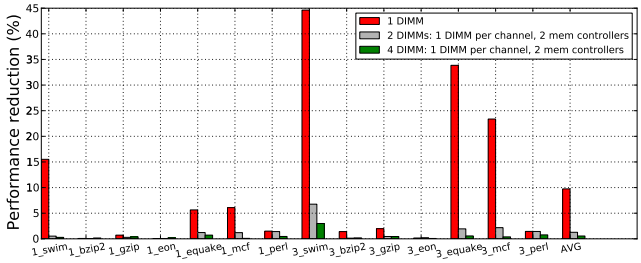


Figure 6. Performance with consolidation

ory pages can fit in the consolidated DIMMs.

To ensure the page migration policy is beneficial, we need to have fast and energy efficient migrations. Figures 7(a), 7(b), 7(c) show the page access patterns of memory and CPU intensive applications. We use M5 a micro-architectural simulator [12], to generate these statistics. We simulate for a representative period of 15 billion instructions of each benchmark. Each graph shows the access distribution for half and full of the simulated interval to examine the changes in the access pattern. These results show that the number of active pages (*hot pages*) is a small fraction of the total number of pages of the application. In general, it is likely we can capture sufficient number of hot pages to migrate during execution quickly enough to resolve thermal emergencies.

4. Joint energy, thermal and cooling management

Our joint energy, thermal and cooling management for memory-CPU subsystems, JETC, is shown in Figure 8(a). It consists of JETC controller, two actuators (memory and CPU) and sensors (temperature and power). On each decision tick the controller analyzes the data provided by the sensors to arrive at appropriate decisions for actuators to implement. The decision takes into account thermal dependencies between CPU sockets and memory. The JETC's controller is implemented in the operating system layer.

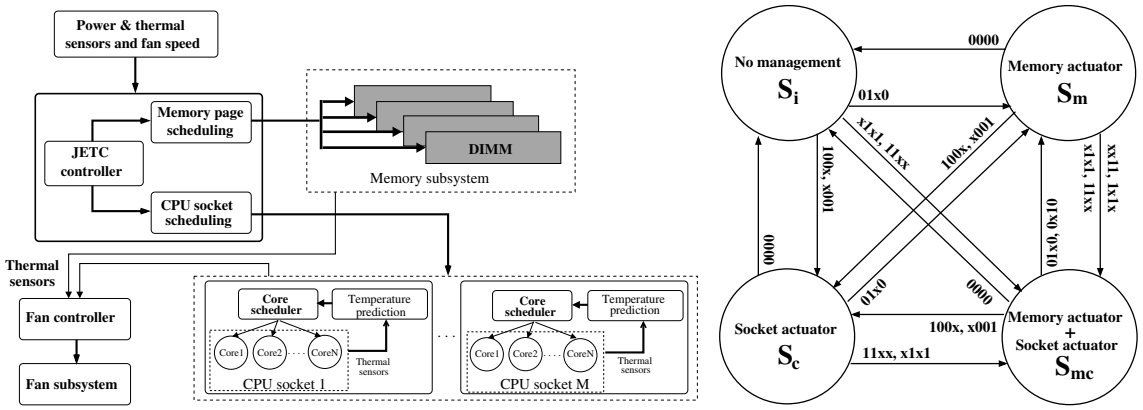
JETC's controller: Figure 8(b) shows the details of JETC's controller. JETC actions are modeled using a state machine with four discrete states: S_{mc}, S_c, S_m, and S_i, to handle four distinct control scenarios of memory, CPU and fan speed. The transitions between states are triggered as a function of fan speed, temperature (CPU and memory) and the number of active DIMMs.

JETC activates only the CPU actuator (**State $\mathbf{S_c}$**) when there are thermal problems in CPU subsystem but no thermal problems in the memory. It activates only the memory actuator (**State $\mathbf{S_m}$**) when there are thermal problems in memory subsystem but no thermal problems in the CPU subsystem. When there are no thermal problems in the system and fans are balanced then JETC does not need to act so it can stay in the idle state (**State $\mathbf{S_i}$**). In this state, no socket or memory page scheduling is performed and the number of active DIMMs is set to the default DIMM clustering size. This default size is defined as the minimum number of active DIMMs (N^D_{min}), in general one DIMM per memory channel to maximize bandwidth.

State $\mathbf{S_{mc}}$: This state handles the cases when there is a need of a joint action from CPU and memory actuators to mitigate thermal problems. The difference in heat generation between CPU sockets can lead to imbalance in speed of fans which is a source of energy inefficiency in the system due to the cubic relation between fan power and speed. This difference in heat generation disturbs the thermal distribution of memory due to the thermal coupling that may lead to imbalance in speed of fans. A transition to S_{mc} happens in any of the following scenarios: 1) imbalanced fan speed with thermal problems in memory (high temperature or number of active DIMMs is higher than default), 2) temperature exceeds threshold in both CPU and memory.

When there is an imbalance in speed of fans, JETC activates the *CPU actuator* to perform socket level scheduling to minimize the difference in heat generation between CPU sockets by migrating workload from a hot CPU to a cooler one. This helps balance the temperature across the memory modules. The CPU actuator also runs core level thermal management to minimize the temperature within the CPUs [9]. At the same time, the *Memory actuator* performs energy aware thermal management by activating just what is necessary of memory modules to meet thermal and performance constraints.

CPU actuator: The details of the CPU scheduler are given in *Algorithm* 1. CPU scheduling can reduce the cooling en-



(a) Architecture (b) JETC's controller

Figure 8. Architecture and controller of JETC. Each edge denotes a transition annotated with a compound condition of 4 bits. Bit-1 (LSB) is true when CPU's temperature is high. Bit-2 is true when memory temperature is high. Bit-3 is true when the number of active DIMMs is larger than the default value. Bit-4 is true when there is a fan speed imbalance. The x-mark denotes a don't care.

ergy of the memory by creating a balanced thermal distribution in memory since CPU and memory are thermally connected. In step 1, the CPU actuator calculates the vector, ΔP^C, of the desired change in power values in each of the CPUs to balance fan speed and to reduce the cooling energy of the memory subsystem. The elements of the vector ΔP^C are computed as follows:

$$\Delta P_i^C = \frac{P_i^C \Delta R_{ca_i}^C}{R_{ca_i} \pm |\Delta R_{ca_i}^C|} \tag{5}$$

where $\Delta R_{ca_i} = (F_i - F_r)\frac{dR_{ca}(F_i)}{dF}$, F_i is the fan speed associated with socket i and F_r is the target fan speed (average of speed of fans in the system). The value of P_i^C corresponds to the socket power and the $(+,-)$ signs in the denominator correspond to the source (hot) and destination (cold) CPUs respectively. The actuator also estimates the power consumed by the individual threads in each CPU by modeling their core and last level cache power that is based on instruction per cycle (IPC), leakage power and cache access rate [8]. The total socket power can be estimated using power dissipation counters in hardware [4].

Our algorithm in steps 2-17 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 on each socket and to reduce the chance of errors. This algorithm removes from a hot CPU (CPU_i) a number of threads that have total power that is less than or equal to $|\Delta P_i^C|$. The cool CPU (CPU_j) receives threads with a total power that is less than or equal to $|\Delta P_j^C|$. Before each migration we evaluate the cooling energy savings to prevent ineffective scheduling decisions. Individual scheduling events are allowed only when the predicted cooling savings are higher than a given threshold, S_{min}. Cooling savings due to the temperature change in CPUs are estimated as in [8]. The temperature change in the DIMMs is calculated using: $\Delta T^D = \lambda \delta P_i^C R_{ca_i}^C$ where δP_i^C is the estimated change in power of the upstream CPU due to thread migration and λ is the thermal coupling factor. The corresponding change in fan speed is calculated using (8).

Algorithm 1 Socket level scheduling

```
1:  Calculate ΔP^C and the set of P^C_thr for each CPU. Set Q as an empty queue
2:  for i in set of hot CPUs do
3:      for j in unmarked threads in CPU i do
4:          dest ⇐ index of coolest CPU
5:          if (P^C_thr_j ≤ |ΔP^C_i| and P^C_thr_j ≤ |ΔP^C_dest|) then
6:              if CPU dest has an idle core then
7:                  Calculate cooling savings of migrating thread j to CPU dest
8:              else
9:                  Calculate cooling savings of swapping thread j with the coolest
                    thread from CPU dest
10:             end if
11:             if cooling savings > S_min then
12:                 Enqueue this migration in Q
13:                 Update ΔP^C and threads assignment
14:             end if
15:         end if
16:     end for
17: end for
18: Execute migration events in Q
```

***Memory actuator*:** This actuator keeps the temperature of memory modules within the safe zone while minimizing energy. It also accounts for fan speed and CPU heat generation since they affect the memory temperature. This actuator may need to either increase or decrease the DIMMs power dissipation using page migration knob when possible. However, when temperature is high and there are no active DIMMs that can accept additional pages, then we choose between activating a new DIMM or letting the fan spin up to cool the DIMMs. We decide between these two options based on the temperature reduction each choice can deliver under the same energy budget. Next we study these scenarios.

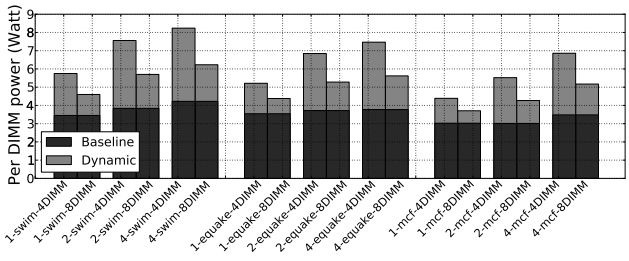


Figure 9. DIMM power breakdown

A. Increasing fan speed versus activating a new DIMM: Figure 9 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. This means that increasing the fan speed may be a better option if it results in a lower temperature at power consumption of P_{base}. Increasing fan speed reduces the memory temperature by lowering its convective resistance as well as the effect of thermal coupling as the temperature of the upstream CPU reduces too. The temperature reduction for increase in fan power by P_{base} can be computed using our thermal model as:

$$\Delta T_{fan} = \Delta F(\lambda P^C_{act}\frac{dR^C_{conv}(F)}{dF} + P^D_{act}\frac{dR^D_{conv}(F)}{dF}) \quad (6)$$

where ΔF is the increase in fan speed of the hottest memory zone. P^C_{act} and P^D_{act} correspond to the CPU and hottest DIMM power that is located in the CPU zone respectively. When the actuator decides to activate a new DIMM, it migrates pages from other DIMMs to the newly activated DIMM progressively from the application starting with the one that has the highest memory access to reduce power density. Using our thermal model, the temperature reduction, $|\Delta T_{mem}|$, for adding one more DIMM can be computed as:

$$\Delta T_{mem} = \frac{(T^D_{jmax} - P_{base}R^D_{ca} - \lambda T^C_{ha})}{n_D} \quad (7)$$

where n_D is the number of DIMMs after expansion and λ is the thermal coupling factor between CPU and memory. The T^D_{jmax} is the maximum temperature of the DIMMs. The controller choses 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:

$$\begin{aligned} &\textbf{if}(|\Delta T_{fan}| < |\Delta T_{mem}|) \Rightarrow \textit{activate a new DIMM} \\ &\textbf{else} \Rightarrow \textit{increase fan speed} \end{aligned} \quad (8)$$

B. Controlling DIMMs power by page migration: When temperature is high in subset of DIMMs, the actuator migrate pages from these DIMMs to the ones that can accepts more pages until the temperature drops to the safe zone. The other scenario is when all active memory modules can accept more pages (cold DIMMs), the actuator migrates pages from the most recently activated module to the rest so that it can be placed in a low power mode when no active pages are left. The actuator maximizes performance by migrating pages to a memory module only if it has lower memory access rate and lower power consumption than the average values. 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. Whenever a page is migrated from DIMM$_i$ to DIMM$_j$, the virtual to physical address mapping for the page is updated in the page table of the operating system. The overhead of migrating pages is acceptable since the difference between the temperature time constant and page migration time is over 6 orders of magnitude.

Table 1. Design parameters

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 (CFM) [3]
	Heat sink time constant at max air flow	25 seconds
	Temperature Threshold	90°C [24]
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 (CFM)
	Chip thermal resistance in K/W	R^D_{chip} = 4.0 [5]
	Heat spreader time constant at max air flow	70 seconds
	Temperature Threshold	85°C [21, 20]
	Self refresh power per DIMM	0.15W
	Thermal coupling factor with CPU	0.65
Fan	Fan power per socket	29.4 W [6]
	Max air flow rate per socket	53.4 CFM [6]
	Fan steps	32
	Fan sampling interval	1 second
	Idle fan speed	10% of max speed

5 Evaluation

5.1 Methodology

In this paper we run the workload in our server and collect real power traces from the memory modules. These power traces are used to estimate temperature using our extended version of the HotSpot simulator [27]. We also measure the CPU core, L2 and baseline power using the method described in [8]. The core and L2 power traces are used to estimate the core and L2 temperatures respectively using HotSpot simulator and Xeon processor layout. We also include the baseline power of the CPU in the temperature simulations.

The memory organization is assumed to be similar to our Intel server (see Figure 1(b)). The memory controller puts the inactive DIMMs in a self-refresh mode and the rest in the active mode. Power consumption during self-refresh is given in Table 1. We use M5 simulator to generate the page access statistics of the applications [12]. Whenever a page is migrated to a different DIMM the virtual to physical address mapping for the page is updated in the page table.

The characteristics of the fan and thermal package of both CPU and memory DIMMs that we use in thermal simulation are listed in Table 1. For the *fan control algorithm* we assume a closed loop PI (Proportional and Integral) controller that is usually used in modern systems. In our setup each fan is controlled individually depending on the temperature of its associated CPU and memory modules. We set the fan to trigger 5 degrees below the thermal threshold of the chip (please refer to Table 1) to allow for enough time to react to thermal emergencies.

For the memory and socket level thermal managers we set the scheduling interval to 4 seconds, since the thermal time constant of both CPU and memory is on the order of tens of seconds. Our technique is not restricted to this interval and other intervals around this value can be used. The interval of fan interval is set to 1 second so as to allow for a detailed control over temperature to ensure reliability. We also set the cooling savings threshold to a conservative value of 10%. For core level thermal management policy, we employ the previously proposed proactive thermal management which perform thermal aware core level scheduling at each OS tick [9].

A set of benchmarks are selected that exhibits various levels of CPU intensity to emulate real life applications (see Table 2). We run each benchmark in the work set till its completion and repeat it until the end of simulation time. The evaluation time for our algorithms is set to 10 minutes after the warm-up period. In our experiments we use a set of workloads that have a representative mix of CPU and memory bound applications. The list of the workload combinations that we use is given in Table 3.

JETC does an integrated energy, thermal and cooling management for CPU and memory. Depending upon the state of the system, JETC takes a coordinated action which is a combination of CPU scheduling and memory thermal and power management. We evaluate JETC against the following set of policies:

Dynamic Load Balancing (DLB): This policy is usually implemented in modern operating systems which performs thread migration to minimize the difference in task queue lengths of the individual cores [14]. The balancing threshold is set to one to make the difference in task queue length equal to zero as possible. In this work we implement the DLB as our *default policy* for the purposes of comparison. This policy keeps all memory modules active. When the temperature of CPU or memory exceeds the threshold, their activity is throttled progressively depending upon the source of the problem. This throttling is included in all the policies.

Dynamic Thermal Management of Core and Memory with PI fan control (DTM-CM+PI): This policy implements state of the art thermal management techniques which operate separately from each other. The core policy forecasts dynamically the temperature of the cores and migrates the threads from the hot to the cooler cores when it is possible to create a more balanced thermal distribution [9]. Memory thermal problems are managed by halting the accesses when the temperature exceeds thermal emergency level until it drops to the safe zone [21, 22]. All memory modules stay active in this policy.

No Fan-Memory Optimization (NFMO): This policy is like JETC except that we disable the actuator that performs the trade off between activating a new memory module and increasing the fan speed. We choose this policy to evaluate the impact of deactivating this optimization on the overall savings. The minimum number of active DIMMs is set just as in JETC.

No Memory Management (NMM): This policy is like JETC except all memory modules remain active the entire time. The purpose of this policy is to evaluate the savings that we can achieved with minimizing thermal hot spots and optimizing the cooling costs in the CPU subsystem only.

No CPU Migration (NCM): This policy disables the CPU management (socket and core scheduling) from JETC. We study NCM to show the impact of deactivating CPU policy on the overall cooling energy savings. The minimum number of active DIMMs is set as in JETC.

5.2 Results

Energy savings: Figure 10(a) show the total energy savings in a system that has 8 DIMMs with 45^{o}C local server ambient temperature. Our energy calculations include total energy of memory subsystem and cooling. JETC delivers savings reaching an average of 50.7% and 25.49% relative to DLB and DTM-CM+PI respectively. The results clearly show that JETC outperforms all other policies. For example, the case of workload W6 includes two *mcf* workload (memory bound), one on each socket, hence they produce comparable amount of heat in each socket and their thermal coupling effect on the memory is comparable. The JETC savings in this case come from clustering the memory accesses to a smaller number of DIMMs since the thermal distribution across the sockets is balanced and there are no thermal problems in the CPUs. The policy NCM performs equally well to JETC since it optimizes for memory and there are virtually no savings opportunities from CPU thermal management. The policies NMM and DTM-CM+PI deliver almost no savings since they do not optimize for memory energy. On the other hand, the savings opportuni-

Table 2. SPEC Benchmarks characteristics

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

Table 3. Workload combinations for multi-tier algorithm

Workload	Socket A	Socket B	Workload	Socket A	Socket B
W1	*equake + 2gzip*	*3bzip2*	W9	*mcf + 2gcc*	*perl + bzip2 + eon*
W2	*2bzip2 + 2eon*	*mcf + gcc + 2gzip*	W10	*3gzip*	*2perl + eon*
W3	*equake + 2bzip2*	*2gzip + gcc*	W11	*2equake + gcc*	*2perl + equake*
W4	*perl + eon + bzip2*	*2equake + gcc*	W12	*mcf + 2gcc*	*2gcc + 2perl*
W5	*2gcc + perl*	*swim + 2gcc*	W13	*gcc + 2perl*	*equake + 2gcc*
W6	*mcf*	*mcf*	W14	*2mcf + gcc*	*2perl + mcf*
W7	*gcc + 2gzip*	*2bzip2 + perl*	W15	*2swim + gcc*	*2perl + swim*
W8	*perl + bzip2 + 2eon*	*2mcf + 2gcc*			

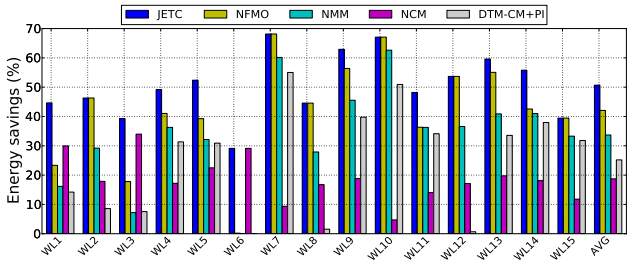
ties for the cases of W7 and W10 are mainly related to imbalance in the thermal distribution between and within the two CPU sockets which raise cooling energy costs. JETC performs quite well since it has the capability to balance the temperature between and within the CPU sockets. The policies NFMO and NMM perform equally well to JETC since they balance socket and core temperatures. On the other hand, DTM-CM+PI delivers a fraction of the JETC savings since it performs only local thermal optimizations. The policy NCM does not perform well since it only lowers memory energy. The other important scenario is when there are savings opportunities for minimizing memory energy consumption and reducing CPU temperature imbalance as shown in W4 and W5.

Figure 10(b) shows the combined memory and cooling energy savings with local ambient temperature of 35^oC. JETC achieved savings reaching an average of 31% relative to both DLB and DTM-CM+PI. In general, the savings are lower than the case of 45^o ambient temperature because most of the savings come from memory clustering as the system requires less cooling. The evidence of this can be seen from the savings of NMM and DTM-CM+PI policies which are close to zero while NCM perform closer to JETC. The JETC policy is able to perform well since it can capture this class of savings opportunities by clustering the memory access to a smaller set of DIMMs. These results also illustrate the benefits of the optimization that trade off between activating a new DIMM and speeding up the fan. The evidence of this benefit can be indicated from the low savings of NFMO since it does not perform this optimization. Since most of the savings come from clustering the memory modules then we expect that the savings would be lower when the workload exercises memory lightly. The cases of W7 and W10 illustrate this scenario.

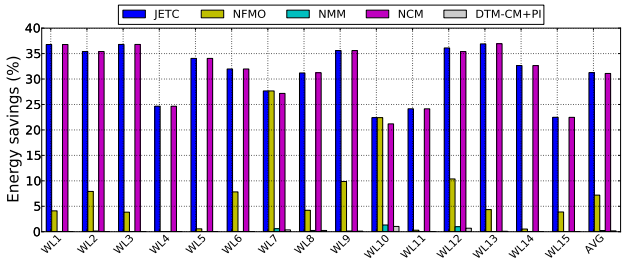
Fan balancing: Next we evaluate the effectiveness of the socket level scheduling in balancing the fan speed to minimize cooling energy where the optimal fan speed ratio between the two sockets equal to 1. Figure 11(a) shows fan speed ratio in a system with 8 DIMMs and 45^oC local ambient temperature. The results show that JETC is able to reduce the standard deviation from the optimal fan speed ratio by 62% compared to the DLB policy. The reason that DLB has a high standard deviation from the optimal fan speed is because it does not account for heterogeneity in the workload in terms of the induced heat in the CPUs. The policy DTM-CM+PI does not balance the temperature between CPU sockets resulting in a smaller effect on balancing fan speed. The fan imbalance with this policy can be even higher than DLB (e.g W8 and W12). This scenario occurs when core level scheduling is effective in one socket while it is not as effective in the other one (e.g. running CPU intensive workload). This increases the imbalance as the speed of one fan can be lowered while the other stays almost the same. The other polices that implements socket level thermal balancing perform close to JETC. Figure 11(b) shows the fan speed ratio in a system with 8 DIMMS and 35^oC ambient temperature. The reduction in standard deviation to the optimal target in this case is 94%.

Page migration: We computed page migration per second to show the robustness of our solution. Figure 12(a) shows the rate of page migration in a system with 8 DIMMs and 45^oC ambient temperature. The results show that JETC average rate of page migrations is below 5 pages per second. This overhead is negligible in practice since migrating a page takes a few microseconds which makes our solution highly attractive. Furthermore, having a few migrations is an indication that our solution is stable. The NFMO has the highest migration rate since it may activate more DIMMs as it does not consider the trade off between fan speed and number of active DIMMs. The case of W15 has the highest migration rate since the workload mix has 3 instances of *swim*, *swim* has a wide set of hot pages. The page migration rate for the system with 8 DIMMs and 35^oC ambient temperature is shown in Figure12(b). In this case, the migration rate for JETC drops below one page per second which causes almost no overhead.

Overhead: Table 4 shows the performance overhead breakdown for the entire set of workloads that includes page

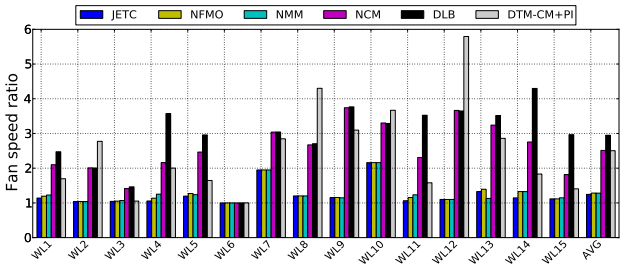


(a) Local ambient temperature is 45^{o}C

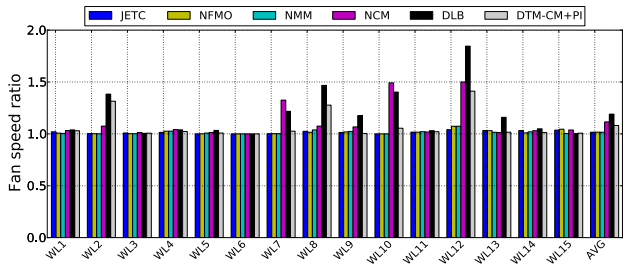


(b) Local ambient temperature is 35^{o}C

Figure 10. Total energy savings (memory+cooling) in a system with 8DIMMs



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



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

Figure 11. Fan speed ratio in a system with 8DIMMs

migration, CPU scheduling (socket + core) and throttling for 45^{o}C and 35^{o}C ambient temperatures. For page and thread migrations we account for both microarchitectural and OS related overheads. The average overhead of JETC at 45^{o}C is 0.27% (includes DIMM clustering overhead that is around 0.2%), which is negligible. The overhead is small due to the big difference between migration cost and temperature time constant. The overhead of JETC is lower than NFMO since the later causes more page migrations and slightly higher CPU scheduling. The policy NCM has lower overhead then JETC since there are no CPU related migrations as it relies on the fans to remove the hot spots from the CPUs. The policy DTM-CM+PI also has a lower overhead compared to JETC since it keeps the DIMMs active which lower the chances of thermal problems in memory and also there are no socket level migrations as it relies on the fan subsystem to manage the heat imbalance between the CPUs. The overhead in DLB is the lowest since it rely on the fan subsystem only to prevents the thermal problems. Although DLB and DTM-CM+PI have lower performance overhead than JETC, the combined energy costs in the case of JETC is much lower than DLB and DTM-CM+PI which easily compensates the slight performance advantage of these policies.

6. Acknowledgment

This work has been funded by NSF Project GreenLight grant 0821155, NSF SHF grant 0916127, NSF ERC CIAM, NSF Variability, NSF Flash Gorden, CNS, NSF IRNC, Translight/Starlight, Oracle, Google, Microsoft, MuSyC, UC Micro grant 08-039, and Cisco.

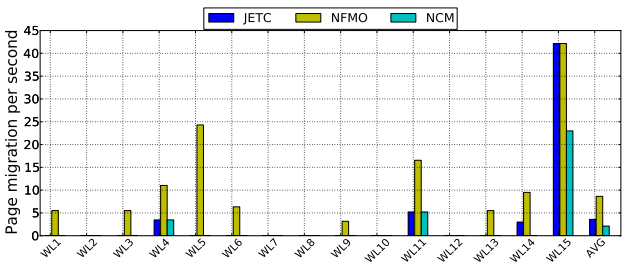
7 Conclusions

Memory and cooling subsystems consume significant amount of energy in high end servers. Many of the proposed techniques focus on power optimizations and ignore cooling costs or ignore the thermal dependencies between CPU and memory resulting in lower energy savings. In this work we have developed JETC, a joint solution that integrates the energy, thermal and cooling management to maximize energy savings. We have designed a unified thermal and cooling model for CPU, memory and fan subsystems. JETC consists of a controller, sensors and actuators (CPU and memory). JETC takes into account thermal and power states of CPU & memory, thermal coupling, and fan speed to arrive at more optimal decisions. CPU and memory actuators can be activated jointly or separately depending on the thermal and cooling state of the system. The memory actuator of JETC reduces the operational energy of memory by cooling aware clustering pages to a subset of memory modules. CPU actuator of JETC saves cooling costs by removing hot spots between and within the sockets, and reduces the effects of thermal coupling with memory. Our experimental results show that employing JETC results in 50.7% average energy reduction of memory and cooling subsystems with less than 0.3% performance overhead.

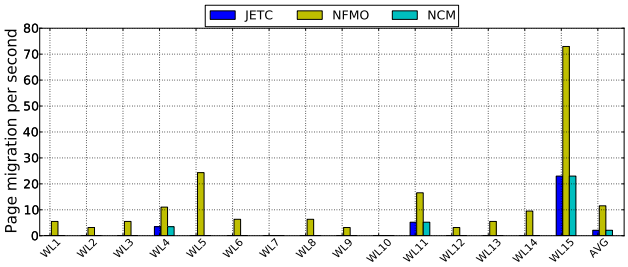
References

[1] *www.sun.com/servers/x64/x4270/.*

[2] *www.intel.com/products/server/motherboards/s5400sf.*



(a) Local ambient temperature 45^oC



(b) Local ambient temperature is 35^oC

Figure 12. Page migration in a system with 8DIMMs

Table 4. Performance overhead (%)

Local ambient temperature	Overhead source	JETC	NFMO	NMM	NCM	DLB	DTM-CM+PI
45^oC	CPU scheduling	0.0552	0.0642	0.0731	-	-	0.0673
45^oC	Page migration	0.0002	0.0005	-	0.0001	-	-
45^oC	Throttling	0.0161	0.0173	0.0254	0.0012	0.0011	0.0022
35^oC	CPU scheduling	0.0152	0.0196	0.0250	-	-	0.0200
35^oC	Page migration	0.0001	0.0007	-	0.0001	-	-
35^oC	Throttling	0.0001	0.0005	0.0011	0.0000	0.0000	0.0000

[3] *Quad-Core Intel Xeon Processor 5300 Series: Thermal/Mechanical Design Guidelines*.

[4] *http://software.intel.com/en-us/blogs/2011/03/30/*.

[5] *www.micron.com/products/dram/*.

[6] *www.sunon.com.tw/products/pdf/DCFAN/PMD4056.pdf*.

[7] A. Ajami, K. Banerjee, and M. Pedram. Modeling and analysis of nonuniform substrate temperature effects on global interconnects. *IEEE Trans. on CAD*, pages 849–861, 2005.

[8] R. Ayoub, K. R. Indukuri, and T. S. Rosing. Temperature aware dynamic workload scheduling in multisocket cpu servers. *TCAD*, 30(9):1359–1372, 2011.

[9] R. Ayoub and T. S. Rosing. Predict and act: dynamic thermal management for multi-core processors. *ISLPED*, pages 99–104, 2009.

[10] L. Barroso and U. Holzle. The case for energy-proportional computing. *IEEE Computer*, 40(12):33–37, 2007.

[11] L. Barroso and U. Holzle. The datacenter as a computer an introduction to the design of warehouse-scale machines. 2009.

[12] N. Binkert, R. Dreslinski, L. Hsu, K. Lim, A. Saidi, and S. Reinhardt. The m5 simulator: Modeling networked systems. *IEEE Micro*, 26(4):52 – 60, 2006.

[13] D. Brooks and M. Martonosi. Dynamic thermal management for high-performance microprocessors. *HPCA*, pages 171–182, 2001.

[14] J. Choi, C. Cher, H. Franke, H. H. A., Weger, and P. Bose. Thermal-aware task scheduling at the system software level. *ISLPED*, pages 213–218, 2007.

[15] A. Coskun, T. Rosing, and K. Gross. Proactive temperature management in mpsocs. *ISLPED*, 2008.

[16] J. Donald and M. Martonosi. Techniques for multicore thermal management: Classification and new exploration. *ISCA*, pages 78–88, 2006.

[17] X. Fan, C. Ellis, and A. Lebeck. Memory controller policies for dram power management. *ISLPED*, pages 129 – 134, 2001.

[18] H. Hai, S. Kang, L. Charles, and K. Tom. Improving energy efficiency by making dram less randomly accessed. *ISLPED*, pages 393–398, 2005.

[19] S. Heo, K. Barr, and K. Asanovic. Reducing power density through activity migration. *ISLPED*, pages 217–222, 2003.

[20] C.-H. Lin, C.-L. Yang, and K.-J. King. Ppt: Joint performance/power/thermal management of dram memory for multi-core systems. *ISLPED*, pages 93–98, 2009.

[21] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang. Thermal modeling and management of dram memory systems. *ISCA*, pages 312–322, 2007.

[22] J. Lin, H. Zheng, Z. Zhu, H. David, and Z. Zhang. Software thermal management of dram memory for multicore systems. *SIGMETRICS*, pages 337–348, 2008.

[23] S. Liu, B. Leung, A. Neckar, S. Memik, G. Memik, and N. Hardavellas. Hardware/software techniques for dram thermal management. *HPCA*, pages 515–525, 2011.

[24] M. Patterson. The effect of data center temperature on energy efficiency. *Proc. ITHERM*, pages 1167–1174, 2008.

[25] M. Pedram and S. Nazarian. Thermal modeling, analysis, and management in vlsi circuits: principles and methods. *Proceedings of the IEEE*, pages 1487–1501, 2006.

[26] D. Shin, J. Kim, J. Choi, S. Chung, and E. Chung. Energy-optimal dynamic thermal management for green computing. *ICCAD*, 2009.

[27] K. Skadron, M. Stan, K. Sankaranarayanan, W. Huang, S. Velusamy, and D. Tarjan. Temperature-aware microarchitecture: Modeling and implementation. *TACO*, pages 94–125, 2004.

[28] Q. Tang, S. Gupta, and G. Varsamopoulos. Thermal-aware task scheduling for data centers through minimizing heat recirculation. *Proc. ICCC*, pages 129–138, 2007.

[29] Z. Wang, C. Bash, N. Tolia, M. Marwah, X. Zhu, and P. Ranganathan. Optimal fan speed control for thermal management of servers. *IPAC*, pages 1–10, 2009.

[30] I. Yeo, C. Liu, and E. Kim. Predictive dynamic thermal management for multicore systems. *DAC*, pages 734–739, 2008.