HeVac: A Heterogeneous Vacation Scheme for Thermal-Aware Multicore Packet Processing

Chih-Hsun Chou
Department of Computer
Science and Engineering
University of California, Riverside
cchou011@cs.ucr.edu

Laxmi N. Bhuyan
Department of Computer
Science and Engineering
University of California, Riverside
bhuyan@cs.ucr.edu

***Abstract*: As processor power density increases, thermal and power control becomes critical for packet processing. Network applications feature ON/OFF execution pattern, which causes frequent temperature and power consumption changes in the processor. In this paper, we propose a novel power aware thermal management algorithm, which achieves power saving in multicore processors by employing a vacation scheme. We implement the scheme through the idle states (C-state) provided by the OS in CPU and show their effectiveness through experimental data. Also, the performance of the vacation scheme is analyzed based on the vacation queuing theory. Then, we propose a heterogeneous load distribution, which creates more opportunities for power saving and temperature reduction. Finally, we design, implement and evaluate an online scheduling algorithm to dynamically choose the best configuration of load distribution and vacation period under real world traffic trace. The technique maintains processor temperature below the temperature constraint and achieves power saving with minimum latency increase. To the best of our knowledge, this is the first work to discuss and develop vacation algorithm considering all power, temperature and latency in the packet processing on a general purpose multicore processor.**

Keywords—Network application; Packet processing, multicore processors, Power saving, Thermal aware technique, vacation and runtime adaptation.

I. INTRODUCTION

The explosive growth of network bandwidth requires orders-of-magnitude increase in packet processing throughput. In addition, many applications (e.g., fast IP-lookup, real-time video streaming, L7-filter, packet classification and deep packet inspection) demand not only high throughput but also low latency. The network industry is aggressively scaling the capability of network processing by employing high-performance off-the-shelf computers to meet this challenge. However, high power consumption in these computers poses a significant challenge to scalable network system design.

Several research prototypes have demonstrated that multicore processors are capable of high-performance packet processing (line rates of 10 Gbps or more) for IP forwarding [1], packet classification [2], deep packet inspection [3], L7 filter [4] and cryptographic operations [5]. These systems not only provide the flexibility to deploy new packet processing algorithms; they also incorporate parallel scheduling to exploit packet-level parallelism on multiple cores for higher throughput and lower latency. Along with increased throughput, however, comes significantly increased power consumption. Collectively, millions of servers in the global network consume a great deal of power [7,22]. Some of the earlier works have proposed vacation and rate adaptation schemes to reduce power consumption in network devices and channels [15-20]. However, none of them focused specifically on packet processing in multicore servers or thermal aware scheduling with temperature constraints.

In this paper, we reveal many interesting relationship between power saving and sleep states (C-states) in the OS in a multicore server. While power consumption remains the major challenge, another factor that must be considered is operating temperature. It is known that high operating temperature adversely affects not only the system performance and leakage power, but also the throughput [8], circuit reliability and chip lifetime [9]. In addition, the cooling and packaging cost for heat dissipation increases exponentially with the power and peak temperature [10]. However, no work has been done on addressing the problem of high operating temperature, as well as the trade-off between thermal constraint, power reduction and packet latency during packet processing.

Network packet processing exhibit an ON/OFF pattern [11,12], where temperature rises during the ON period and falls during the OFF period. The short idle periods prevent the CPU cores from entering deep idle state (C-state), provided by modern CMP OS, and thus consume high idle power. In this paper, we propose a thermal/power management *Vacation* scheme, which forces more idle period and reshapes packet processing pattern to create opportunities for a PE to enter power saving mode (deeper C-states). Also, we put a limit on the length of busy period, which prevents cores from overheating. By adopting our proposed *vacation* scheme, we can reduce the power consumption during idle periods, and decrease the peak temperature. Then, we propose heterogeneous load distribution which creates more opportunities for long vacation period and achieve even better power saving and temperature reduction.

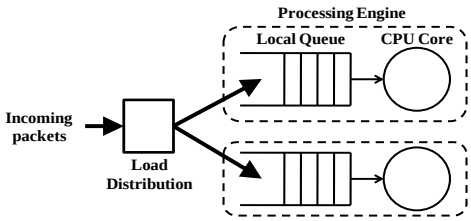

Figure 1. Classical multicore server.

Finally, we propose ***HeVac***, a thermal and power aware **He**terogeneous **Vac**ation runtime, which measures the packet intensity and predicts the power consumption and temperature using analytical techniques, derived in this paper. Then it automatically distributes different amount of load among cores to achieve power saving without violating a predefined thermal constraint. Besides, the scheme provides maximum power saving under traffic variation. To the best of our knowledge, *HeVac* is the first to develop a vacation scheme considering the trade-off between power and temperature in the network packet processing on a multicore server.

This paper makes the following contributions. (1) We implement a vacation scheme through the idle states (C-states) provided by the OS and observe the packet processing characteristics under different traffic load. (2) We derive analytical models for power, thermal and latency characteristics for the vacation scheme based on the vacation queuing theory. (3) We explore the effect of power saving and temperature reduction with different load distribution. (4) We design, implement and evaluate ***HeVac***, an on-line algorithm, to dynamically choose the proper power/thermal management and load distribution based on the traffic variation to achieve power saving under a temperature constraint.

II. Packet Processing on a Multicore Server

This section describes the high-level network application characteristic, and relates it to the power and thermal behavior while processing on a multicore processor. Also, we study the default operations to the C states in the OS.

A. Packet Processing Characteristics

The "run-to-finish" applications always consume active power while running, and the temperature simply rises to saturation point until it stabilizes. But network packet processing exhibit an ON/OFF pattern [11,12], where the CPU core temperature rises and consumes active power during the ON period, (also known as *busy period)* when the packet is being processed. During the OFF period when no packet arrives, the CPU still runs and consumes idle power which is less than the active power. The OFF period in-between packet processing provides the opportunities to do power and thermal management. Also, the length of ON/OFF period is not deterministic, and depends on the packet arrival and service rates.

We start by considering how packets arrive and are served. Fig. 1 shows the general packet processing system in multi-

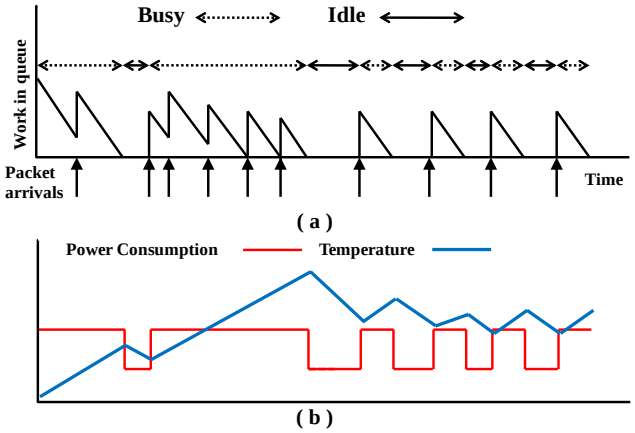

Figure 2. (a) Packets processing pattern. (b) Power consumption and thermal behavior during packet processing.

core architecture; we use two-core processor as an example. The system consists of two processing engines (PEs); each PE has a local queue and a CPU core in our case. The server receives packets through NIC following either TCP or UDP protocol. In this paper, we will focus on TCP only. Incoming packets are distributed among PEs through load distributor. The packets arriving at each PE are stored in its local queue, and the CPU core constantly checks whether the queue is empty or not. If the queue is not empty, CPU core will fetch and process a packet in the queue in an FIFO fashion driving the CPU to the active state. On the other hand, while the CPU core finds no packet in the queue, it will become idle (entering idle state) until the next packet arrives and is stored in the queue.

B. Power Consumption and Thermal Behavior

Let us now discuss how the power consumption and temperature will behave in a CPU-based packet processing system. Fig. 2a depicts the packet processing pattern of a PE on a classical multicore server, shown in Fig.1. The y-axis is the amount of work presented in the local queue of a PE, and x-axis is the time. The solid lines represent the amount of work in PE queue with time. The solid up arrows under the x-axis represent the packet arrivals to the PE, which cause the step increase of the work in the queue. The slant line represents the packet being processed by a processor with the slope representing the service rate. The busy and idle period are shown in the top of the figure. We can clearly see the busy and idle periods during packet processing are randomly distributed.

Fig. 2b shows power consumption and thermal behavior with the same packet arrival pattern. We can see that when PE is active, they consume more power, and drop to a lower level when idle. Besides, one should note that the lengths of busy and idle periods are randomly distributed with the expected lengths of $\frac{1}{\mu-\lambda}$ and $\frac{1}{\lambda}$, where, λ is the packet arrival rate and μ is the service rate of a PE. Then the average power consumption of a PE can be formulated as:

$$P_{avg} = \frac{P_a \times t_a + P_s \times t_s}{t_a + t_s} = P_a \times \rho + P_s \times (1-\rho) \quad (1)$$

where P_a is the active power consumption, P_s is the idle power consumption, t_a is the length of busy period, t_s is the idle period and ρ is the traffic load.

For the thermal behaviors, we can see that the temperature trace is like the shape of sawtooth, which is caused by the continuous busy and idle period. The longer the busy period, the higher the temperature will rise until a steady state. On the contrary, the longer the idle period, the lower the temperature will fall. Also, due to the heat flows between CPU cores in a multicore architecture, their temperature behavior will interact with each other. The complicated thermal behaviors pose a major challenge for the temperature management.

C. Interrupt handler and TCP/IP protocal stack

In the real-world packet processing system, the packets processing capability will vary among cores. In addition to different packet lengths, hardware interrupts, generated by the incoming traffic, and TCP/IP protocol stack also present asymmetrical load on the PEs.

When a message is received, it needs to go through the normal TCP/IP protocol stack. The interrupt handler is responsible to receive the message from NIC, and then copy them to kernel buffer; at the end application layer reads the message data to user application space by system calls. The packet processing performance will degrade because of the cost of interrupt and TCP/IP protocol stack.

Fig. 3 shows our experimental result of the service rate with and without the cost of interrupt. In this experiment, all four cores of our quad-core machine are processing packets, but PE0 alone is handling the interrupt and TCP/IP stack. We can see that as the arrival rate increases, the service rate of PE0 is decreased linearly, and PE1 to 3 are unaffected. The service rate under the influence of interrupt cost can be modeled as:

$$\mu_e = \mu_0 - 0.035\lambda \quad (2)$$

, where μ_e is the affected service rate, μ_0 is the uninterrupted service rate, and λ is the total arrival rate to the system.

Let the achievable packets service rate for each PE be μ. For PE0, while receiving interrupts, the service rate will be $\mu_0 < \mu = \mu_{1\text{-}3}$ for packet processing. When the incoming traffic is evenly distributed, the load of each PE will be uneven, $\rho_0 > \rho_{1\text{-}3}$. In order to maximize throughput and avoid larger delays from overused cores, several load balancing schemes can be applied, e.g. give different traffic load to each PE so that $\frac{\lambda_0}{\mu_0} = \frac{\lambda_1}{\mu_1}$, or balance the interrupt handler among cores by using *irqbalance* feature provided by Linux kernel. However, considering temperature and power consumption, balancing load may yield to non-optimum thermal and power efficiency.

III. Thermal Aware Power Management

In this section we first propose a vacation scheme, where the CPU core goes on a vacation (idle state) for a fixed period and then wakes up to serve the packets in the queue.

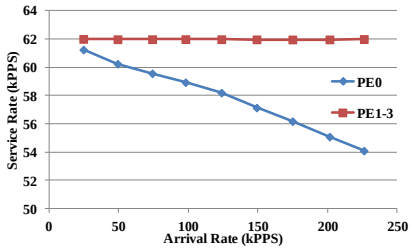

Figure 3. The effect of interrupt handling cost. (kPPS is kilo packets per second)

Also, the server will take a vacation as soon as it finds no packet in the queue. Then, we derive the performance behavior under multicore architecture by applying vacation queuing server model. Lastly, we propose a heterogeneous load distribution to achieve further power saving and temperature reduction on top of vacation scheme.

A. Thermal Management and Power Saving through Vacation Scheme

In the original system, servers are busy processing packets as long as there are packets in the queue, and become idle when the queue is empty. Although servers spend most of their time idle when the traffic load is low, conventional thermal management techniques (clock and power gating) are unable to exploit these short idle periods due to the high timing overhead of changing C states. We propose an approach to reshape the packet processing pattern that creates longer idle period and allows the processors to enter low-power state which consumes less power. We call this approach *vacation*.

Fig. 4a shows the flow chart of the *vacation* approach. Here, we define two states: vacation and working state. In working state, the CPU core will fetch packets from the local queue and process them. The CPU core will switch to vacation state under two circumstances: 1) the busy period exceeds a predefined time period t_{work}, and 2) the CPU core finds no packet in the queue. The time period t_{work} limits the length of the busy periods, preventing the core from exceeding the given temperature threshold. In the vacation state, the server takes a vacation for time t_{vac}, during when the core will buffer all the incoming packets in the queue. After t_{vac} expires, the core wakes up and goes back to the working state, and continues serving packets in the queue.

The effectiveness of the vacation scheme relies on the capability of CPU core to save power when it is idle. Modern CPUs provide succession of idle states (called sleep states or C-states) to save power. States are named numerically from C_0 to C_N , where C_0 represents the active state. As the C-state number increases, the power consumption of the CPU core reduces and response latency increases. Table 1 shows the C-states of our quad-core machine. Response latency of processors increases as we move to deeper sleep states. It only makes sense to enter a C-state if inactive time is beyond certain threshold because of the response latency.

Now, let's discuss the logic behind our proposed vacation scheme. In our vacation approach, the incoming traffic,

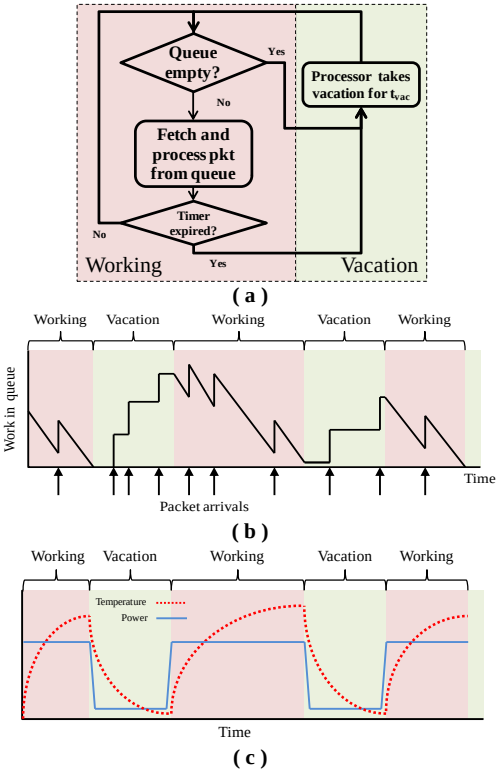

Figure 4. (a) Flows chart, (b) Packets processing pattern, and (c) Temperature and power consumption of vacation scheme.

form core's perspective, is reshaped into short bursts of arrival. Cores wake up to process bursts of packets, and then sleep for a long time. The intent is to provide sufficient time for cores to enter the deeper sleep state with lower power consumption, and temperature cool down. Fig. 4b shows the amount of work of a PE presented in its local queue while time advances. Compared with Fig. 2a, we can see that the amount of work increases due to the queuing during the vacation. In the working state, instead of many short packet processing time periods, it's now a long busy period.

The power and thermal behaviors are shown in Fig. 4c. Several competing factors influence the efficiency of this approach, where the deeper idle states during vacation decrease the power consumption, and cause the temperature to cool down faster. However, since the incoming packets during vacation are "packed" into working state, results in increasing average power consumption and temperature rising during the working period. This tradeoff must be carefully considered during the design.

B. Analysis of the Vacation Period

In order to understand how the performance will be affected by these trade-offs, we propose a performance estimation model by applying vacation queuing theory [6].

Table 1.CPU core C-states.

State	Response Latency	Threshold	Power
C0	N/A	N/A	7.7W
C1	1µs	1 µs	3.4W
C3	59 µs	156 µs	1.9W
C6	80 µs	300 µs	0.5W

Power Model: Since the vacation scheme only reshapes the packet processing pattern, it does not introduce extra idle/busy time. We can apply equation (1), and substitute Ps with Pc, the power consumption of the C-state. Thus, the average power consumption under vacation scheme is:

$$P_{vac\_avg} = P_a \times \rho + P_C \times (1-\rho) \quad (3)$$

, where ρ is the load for each PE.

Temperature Model: We adopt the well known thermal RC model for a CPU core [10]. Let *V(t)* and *K(t)* is the temperature in the vacation and working period at time t. Although the temperature will not converge to steady state, we could derive the thermal behavior in the equilibrium state, by letting $V(t_{vac})=K(0)$, and $V(0)=K(t_{work})$. Also, since the length of both periods, less than 1 ms, are much shorter than the time constant of the pair of resistance and capacitance, about 10 ms [10], we can approximate the thermal behavior by taking the first two terms of its Taylor expansion. The maximum temperature, T_h, can be derived as:

$$T_h = K(t_{work}) = P_C R + \frac{R^2 C}{t_{vac} + RC\rho}(P_{work} - P_C) \quad (4)$$

, where R and C are the thermal resistance and capacitance of the CPU core from our model.

Latency Model: since the working period will either end when the queue becomes empty or timer expires, the system is equivalent to a *single-server queue with vacation and non-gated time-limited service* [6,13]. We propose to use *mean value analysis* to analyze the packet latency in vacation approach. We assume that the working period time limit is large enough to avoid packet being deferred to the next working period. Under this assumption, we can derive latency as

$$E(W) = \frac{E(S_r)}{(1-\rho)} + \frac{t_{vac}}{2} \quad (5)$$

C. Vacation under Multicore Architecture

In the previous section, we discussed the performance of vacation approach under single core architecture. In this section, we will discuss the power consumption, temperature and latency for a multicore architecture, where multple cores process packets at the same time.

Since every core processes packets and takes vacation independently, the power consumption and latency equations will be independent among cores. As a result, the overall power consumption is simply the summation over all cores, and the overall latency is the weighted sum of the latency of every core, where the weight will be the ratio of the load for the core to the overall incoming load.

The thermal behavior is more complex compared with the power consumption and latency. The heat dissipation and temperature difference among cores play a major role. Fig 5a shows the overview of a two-core CPU package. It is composed of a heatsink, which helps heat to dissipate into

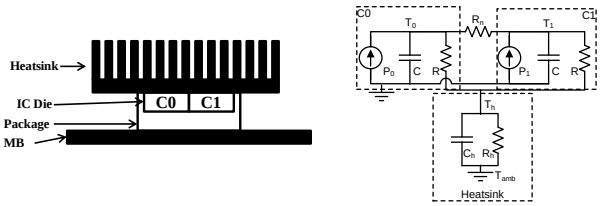

Figure 5. (a) The CPU package overview and (b) the extended RC model of a two-core processor.

the surrounding air, and a CPU package, which protect the two-core IC die. MB is the abbreviation for mother board. When the processor is active, both core0 (C0) and core1 (C1) will generate heat. The heat will flow in two directions: vertically, from core to heat sink, and horizontally, from one core to the other. These heat flows make the thermal behavior of the core not being independent among cores in the multicore architecture.

We develop a thermal model by extending the single core RC model [10] as shown in Fig 5b. The core can be modeled as a current source connected with a pair of resistance and capacitance in parallel. By solving a set of ODE from our model, we can have the thermal behavior of a core as (We have omitted detailed derivations for brevity):

$$V^i(t) = P_C^i R \frac{m+1}{m+2}\left(1-e^{-\frac{m+1}{m+2}\frac{t}{\alpha}}\right) + V^i(0)\left(\frac{1}{2}e^{-\left(\frac{m-2}{m}\right)\frac{t}{\alpha}}\right) + P^n R \frac{1}{m+2}\left(1-e^{-\frac{1}{m+2}\frac{t}{\alpha}}\right) + V^n(0)\left(\frac{1}{2}e^{-\left(\frac{m+2}{m}\right)\frac{t}{\alpha}}\right) \quad (6)$$

, substitute P_c with P_{work}, we will have the working state thermal behavior $K^i(t)$. In Eqn. 6, the superscript i and n represent the modeled core and its neighbor core, P^n is the average power consumption of the neighbor core, $m = \frac{R_n}{R}$, and $\alpha = RC$ is the time consstant.

Following the same principle of deriving Eqn. 4, we can have the maximum temperature, T^i_h, as

$$T_h^i = \left(P_C^i R + \frac{R^2 C}{t_{vac} + RC\rho^i}\left(P_{work} - P_C^i\right)\right) \times \frac{m+1}{m+2} + P^n R \times \frac{1}{m+2} + P^{pkg} R^h \quad (7)$$

, where P^{pkg} is the total power consumption within the CPU package. The first two terms depict the temperature of the core, and the third term is the temperature of the heat sink.

With the multicore thermal behavior in mind, we now discuss the effectiveness of vacation scheme in multicore architecture. The vacation scheme can help reduce temperature in core and package levels. In core level, the vacation period provides a continuous idle period for the core to cool down. Also, when the core enters deeper C-states, the core will cool down to a lower level, as we discussed in the previous section. In package level, the power saving brought from entering C3/C6 from all cores will reduce the total package power consumption and the heatsink temperature, as a result, the temperature of every core will decrease. The experimental results and analysis of the multicore vacation scheme will be presented in Section IV.

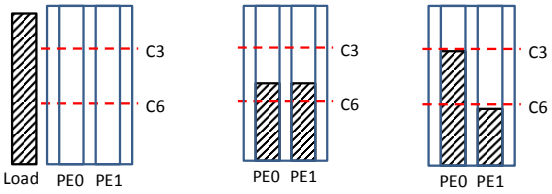

Figure 6. Heterogeneous load distribution.

D. Hetrogeneous Load Distribution

Until this point, we assume that the incoming traffic is evenly distributed among cores. However, this is not the optimum load distribution considering temperature and power consumption. We will give an example about how heterogeneous load distribution can help reduce temperature and power consumption. Fig. 6a shows the two cores system, PE0 and PE1, with incoming traffic load. The red dotted lines represent the boundary of different C-state; if the load for each core does not exceed the C-state boundary line, the core will enter that C-state during its vacation period. Fig. 6b shows the evenly load distribution, we can see that both loads of PE0 and PE1 are beyond C6 boundary but below C3 boundary line, which means that both PE0 and PE1 can enter C3 state during vacation.

Although the CPU can achieve good temperature and power consumption reduction with this normal vacation scheme, we can achieve even better performance with the following observation. The residency time that each C-state needs to be efficient is discrete, e.g. 156μs for C3 and 300μs for C6, which means that the core can enter C3 when the length of vacation period falls between 156 and 300μs. To achieve further power saving, we can migrate part of the load that exceeds C6 boundary from one core to the other core, and the other core can still satisfy the C3 boundary. We name this approach *heterogeneous load distribution*. Fig. 6c illustrates the heterogeneous load distribution, where PE0 enters C3 state and P1 enters C6 state during its vacation period. As a result, the power consumption is reduced by 1.4W (Table 1).

The heterogeneous load distribution achieves better power saving, since one core will enter deeper C-state. However, the effectiveness of temperature reduction relies on the tradeoff between core and heatsink temperature. In Fig. 6c, the PE1 temperature will be lower than that in Fig. 6b, but the PE0 temperature will rise because of the shorter vacation period. The heatsink temperature will decrease because of the lower total power consumption within the package. The experimental results and analysis of the heterogeneous load distribution will be presented in Section IV.

IV. PERFORMANCE EVALUATION

In this section, we present the experimental results and provide some insight on how to effectively use the approaches in practice. We evaluate the performance of vacation and heterogeneous load distribution approaches from the following aspects: (1) how are the temperature, power and latency affected by different vacation period? (2) How is

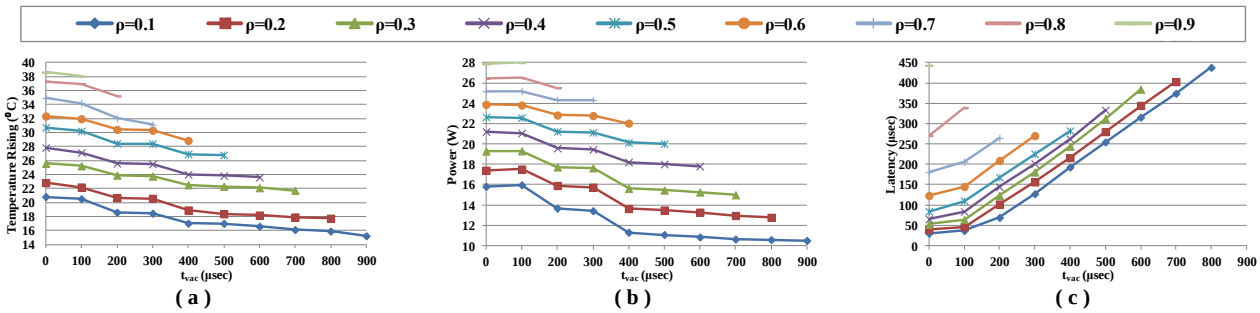

Figure 7. (a) Temperature rise, (b) power consumption , and (c) latency with vacation scheme.

the performance affected by different load distribution? We will elaborate our evaluation methodologies of the experiments in the rest of this section.

We use two multicore machines connected over Ethernet, one to generate packets and the other to process them. We are mainly interested in the behavior of the packet processing server. The server is an Intel i7-3770 quad-core processor. The Intel Hyperthreading, Turbo boost, and Enhanced C1 state (C1E) are disabled through BIOS setting and the machine specific register (MSR). The CPU fan speed is set to be fixed at 1000 RPM. Power consumption and temperature readings are collected from the MSR. Our experiments in his section are based on synthetic traces generated by our client with varying traffic rates. The service rates are obtained from executing real network applications. We will present some results with real-world network traces in the next section.

A. The Effect of Vacation Period

In the vacation approach, the design parameters which will affect the packet processing performance are the lengths of working period (t_{work}) and vacation period (t_{vac}). Let t_{cycle} be the cycle time, defined as $t_{cycle} = t_{work} + t_{vac}$. t_{work} is chosen to satisfy the stability condition : $\frac{t_{vac}}{t_{cycle}} < 1 - \rho$.

In Fig. 7, we vary t_{vac} from 0 to 900 μs (x-axis), where t_{vac} = 0 represents the default scenario without applying our approach. Also, the incoming traffic varied from ρ = 0.1 to 0.9, representing the throughput of 24711 to 225480 packets per second. Furthermore, the incoming traffic is evenly distributed among cores. The detailed discussions are presented below.

Temperature:

Temperature rise is defined as the difference between the idle temperature and active temperature. In these experiments, we first let the all four cores idle for 300 seconds and measure the temperature. Then we run an application CRC for 300 seconds and record the temperature reading every 10ms. We take average of the temperature of last 60 seconds and four cores for every run. From Fig 7a, we can see that the temperature decreases as t_{vac} increases. The temperature cool-down brought from continuous and deep idle state is more significant. The reason is that under low traffic load, there exists a large amount of short idle period. These short idle periods are packed into a long vacation period, and allow the processer to have enough time to enter deeper C-state. On the other hand, under high traffic load, the idle period is still not long enough because that the working period has to increase to satisfy the stable condition. At t_{vac} = 200 and 400μs, the temperature reduction increases because of the entry into C3 and C6 state, respectively. Also, when ρ increases, load during the working period increases, which causes increase of the temperature, and the decrease of the range of t_{vac}.

Power Consumption:

The power consumption results are given in Fig. 7b. We can see that it achieves more than 20% of power saving under low traffic load (ρ<0.4), and 30% when ρ=0.1. However, the effect of power saving is diminished when traffic load is high. We can also observe this through t_{vac}. When t_{vac} is smaller than 200 μs, we can see that the power saving is insignificant, because the short t_{vac} will stop the processer from entering deep C-states (C3 and C6). Also, we can notice that the power consumption drops significantly at t_{vac} = 200 and 400 μs, corresponding to the entrance of C3 state and C6 state in the vacation period.

Latency:

The down side of introducing forced vacation period is that it will increase the average packet latency. However, the latency for processing a TCP flow will not increase very much. Fig. 7c shows that the individual packet latency increases almost linearly as t_{vac} increases. There are two factors that will cause this latency to increase. First, when t_{vac} is larger, which means longer vacation period, the packets arriving during vacation will have to wait for longer time for processing. Second, the packets arriving during the working period will have to wait for the packets already in the queue due to vacation to be processed. The longer the vacation period, the more packets are in the queue, thus longer is the waiting period.

C-states residency:

To further show that vacation approach achieves power saving through the use of processer C-states, we record the time that the processer resides in different C-states. Fig. 8 shows the C-states residency for ρ = 0.3 while t_{vac} is varied from 0 to 700, the total idle time is 206 seconds. We can see that when t_{vac} = 0, processor spends more than 95% of the idle time in C1 state, the least power saving mode. When t_{vac} increases to 200μs, the cores will enter C3 state during vacation period. When t_{vac} further increases beyond 300, cores will enter C6 state and achieve higher power saving because there is long enough vacation period. Compared with the temperature and power consumption shown

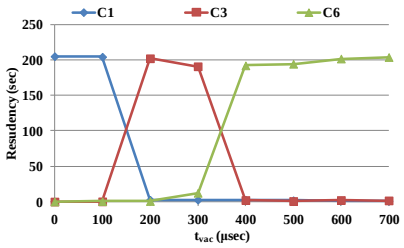

Figure 8. C-states residency with vacation scheme .

in Fig. 7a and 7b, the readings drop significantly t_{vac} = 200 and 400μs, corresponding to the cores entering the C3 and C6 states.

B. Effect of Heterogeneous Load Distribution (HLD)

In this section, we will discuss the effectiveness of our proposed HLD scheme. In this section, we restrict the range of t_{vac} from 0 to 500 μs. Increasing t_{vac} over 500 μs only marginally improves the temperature reduction and power consumption, but significantly degrade the latency performance (Fig. 7). All four PEs are active but the interrupt handling is bound to PE0. Table 2 gives the experimental results before and after applying HLD, including the load distribution among cores, overall power saving, temperature reduction and latency. In this experiment, we vary the incoming traffic load to the system from ρ=0.4 to 0.7. We didn't show the result for other incoming traffic load because when ρ < 0.4, the even distribution will give the best performance without HLD. On the other hand, performance of HLD is limited when ρ > 0.7.

The results of power saving and temperature reduction (Temp. Red.) are compared with the default case where no thermal management technique is applied and load is evenly distributed among cores. For every traffic load, we give the performance under four different distributions: *Even* represents evenly distribution with our vacation scheme; *Case1*, *2* and *3* show the result applying vacation scheme, and HLD with different load distributions. In case1, we migrate some load from PE3 and evenly distribute among PE0-2. In case2 and case3, we further migrate some load from PE2 and PE1 to PEs with smaller index number. The per-PE load and deepest C-state that PE1 to PE3 enter is also given. Because of the interrupt handling, PE0 will be interrupted even during its vacation period, which wakes up the core from idle state. As a result, it hardly enters C-states below C1 and is not shown in the table. Thus, we only apply our vacation scheme to PE1 through PE3.

From Table 2, we can see that by applying vacation scheme itself, we can achieve about 13% power saving and 11% of temperature reduction for ρ = 0.4. After applying our proposed HLD, the performance is further improved to 25% for power and 20% for temperature. At low load, as we migrate more and more load to PE0, the performance of PE0 will decrease. However, since we don't apply vacation scheme in PE0, the performance degradation is not significant compared with the performance gain of the other three PEs. However, the temperature reduction will favor even distribution when the load is high. As load increasing, the heatsink temperature reduction, brought from package power saving, is decreasing and can't compensate the temperature increasing, caused by even higher load for some cores after HLD. As a result, evenly load distribution will have better performance in terms of temperature. Since the performance depends on the length of vacation and the load of each core, equation (1)-(7) are needed to dynamically choose the best configuration for different traffic. We will present our runtime scheduling algorithm with real world traffic trace in the next section.

V. HeVac with Traffic Variation

A. System Overview

In this section, we present an online thermal aware runtime algorithm, *HeVac*, which dynamically applies our vacation scheme and heterogeneous load distribution with varying incoming traffic. Our goal is to achieve power saving or satisfying latency requirement without violating temperature constraint. Fig. 9 shows the system overview of *HeVac*. Our thermal aware packet processing system composed of a scheduler and many PEs. The arriving packets will be sent to the scheduler, and then to the designated PE assigned by the scheduler. The processing core in the PE will fetch and process the packets stored in its local queue It is regulated by the vacation manager, which applies the vacation scheme to the core.

The aim of the scheduler is to distribute the workload to every PE without violating the thermal constraint. The scheduler consists of two parts: performance estimator and load distributer. The performance estimator will first calculate the shortest vacation period needed to prevent thermal constraint violation when the load is evenly distributed among cores, and will be used as a sanity check after the HLD and VAC are applied. The load distributer will migrate load to create the opportunity for power saving through deeper C-state. The detailed runtime algorithm will be presented in the next subsection.

Table 2. HLD with different incoming traffic load (ρ).

	ρ=0.4				ρ=0.5				ρ=0.6				ρ=0.7			
Dist.	Even	Case1	Case2	Case3	Even	Case1	Case2	Case3	Even	Case1	Case2	Case3	Even	Case1	Case2	Case3
ρ_0	0.4	0.432	0.5	0.7	0.5	0.566	0.7	0.8	0.6	0.7	0.75	0.9	0.7	0.732	0.8	0.832
ρ_1	0.4	0.432	0.5	0.3	0.5	0.566	0.7	0.6	0.6	0.7	0.75	0.9	0.7	0.732	0.8	0.832
ρ_2	0.4	0.432	0.3	0.3	0.5	0.566	0.3	0.3	0.6	0.7	0.6	0.3	0.7	0.732	0.6	0.832
ρ_2	0.4	0.3	0.3	0.3	0.5	0.3	0.3	0.3	0.6	0.3	0.3	0.3	0.7	0.6	0.6	0.3
C-states	{C3,C3,C3}	{C3,C3,C6}	{C3,C6,C6}	{C6,C6,C6}	{C3,C3,C3}	{C3,C3,C6}	{C1,C6,C6}	{C3,C6,C6}	{C3,C3,C3}	{C1,C3,C6}	{C3,C3,C6}	{C1,C6,C6}	{C1,C1,C1}	{C1,C1,C3}	{C1,C3,C3}	{C1,C1,C6}
Power Sav.	13.2%	17.3%	21.4%	**25.5%**	10.1%	13.3%	13.1%	**16.4%**	7.5%	4.8%	7.4%	**9.7%**	4.5%	6.1%	7.3%	**7.8%**
Temp. Red.	11.1%	14.1%	17.1%	**20.1%**	15.0%	17.3%	**19.7%**	17.2%	12.2%	**13.8%**	12.1%	10.2%	**8.4%**	7.1%	8.0%	5.7%
Lat.	157.75	154.76	150.5	**145.37**	176.75	171.243	**163.75**	168.9	227	219.79	221.96	**211.62**	311.5	313.225	317.83	**302.67**

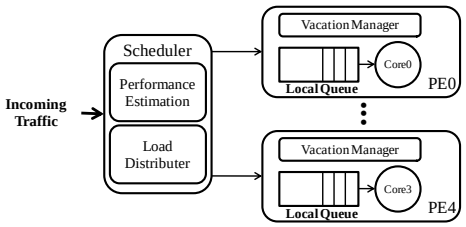

Figure 9. HeVac overview.

B. Runtime Scheduling Algorithm

In the previous section, we observed that the effectiveness of the vacation approach depends on the load and the vacation period of the PE. Algorithm. 1 shows our scheduling algorithm. At the beginning, the incoming load is evenly distributed among 4 PEs, PE1 to 3 will apply vacation scheme, while PE0 remains default and will handle interrupt (LINE 1). First, the t_{vac} for each core is set to the maximum length that will satisfy stability condition, and the corresponding performance will be calculated through Eqn. (3,5,7) (LINE 2-3). Since PE0 is not taking vacation, and is more loaded than the other cores, we need to check if the thermal constraint is satisfied. If PE0 temperature will violate the constraint, we migrate part of its load and evenly distribute to the other cores, then go back to the first step. The amount of load to be migrated can be calculated by reversing Eqn. (7), with P_C equal to C1 state power consumption and t_{vac} = 0. Also, we need to make sure the thermal constraint can be satisfied after applying vacation scheme. This is done by comparing only the T_h of PE1 with the constraint, since PE1 to 3 have identical load distribution and vacation period. If it violates the constraint, other thermal management techniques need to be applied to maintain the temperature under the constraint, such as increasing fan speed. If none of the above two scenarios happened, it then will go to the load distributor. In the load distributor, we follow the same steps as described in case 1, 2 and 3 in Section IV-B. The amount of load we migrate should be able to allow that core to enter one C-state deeper than it originally could. For example, if PE3 originally can enter C3-state, we migrate some load so that it can enter C6-state. Also, we will compare and record the best configuration during these steps. At the end, the recorded configuration will be applied by the vacation manager within each PE.

C. Effectiveness under real-world traffic

In this section, we evaluate the effectiveness of HeVac under real-world traffic. We use real network traces from CAIDA's equinix monitors [14] as inputs. CAIDA monitors capture the traffic traces on OC192 links in real time. The traces in this data set are of one day long duration and were captured in year 2011. The network traces are replayed with actual packet arrival rate in our client machine to model the real traffic scenario.

The experimental data for the power saving under temperature constraint is shown in Fig. 10. In this experiment, we set the temperature constraint as 74°C, which is 5°C under T_{max}. T_{max} is defined as the maximum temperature when all

Algorithm 1. HeVac runtime algorithm.

Init: $\rho^i = \frac{\lambda}{4\mu_i}$, T_c**:thermal constraint,** t_{cycle} **= 1ms**

Performance Estimation:

1: $t^i_{vac} = t_{cycle} \times \rho^i$ **for** $0<i<4$, $t^0_{vac} = 0$.
2: calculate P^i, T^i_h, L^i, *and* $P^{pkg}=\sum P^i$ *for* $0 \le i < 4$.
3: **IF** ($T^0_h > T_c$ & $T^1_h \le T_c$)
4: Migrate load form PE0 and evenly distribute to PE1-3, update ρ^i , **GOTO** LINE 1 and disable HLD.
5: **ELSE**
6: **GOTO** Load Distributor **IF** HLD is not disabled.
7: **END**

Load Distributor:

8: **FOR** (j=4;j>0;j--)
9: Migrate load test: estimate ρ^i, t^i_{vac}, P^i, T^i_h *and* L^i , *and* P^{pkg} after migration.
10: **IF** ($T^0_h > T_c$ *or* $T^1_h > T_c$)
11: DO NOT Migrate load, and **BREAK**
12: **ELSE**
13: **IF** (*perf* > *prev_perf*)
14: config = cur_config
15: **END**
16: **END**
17: **END**
18: **APPLY config to the Vacation Manager**

four cores are constantly processing packets without any idle period. In Fig. 10, the black line represents the load variation on our multicore server, as obtained from the CAIDA equinix-chicago trace. Red line corresponds to the power saving from HeVac. We also give the power saving from only vacation scheme, shown as VAC, in blue line. First, we can see that VAC and HeVac behave the same when $\rho \le 0.4$, because PE1~3 can enter C6 state with even distribution while load is low, there is no need to apply HLD. When $\rho > 0.4$, HeVac outperforms VAC, since it allow some PEs enter deeper C-states. Also, Fig. 10 shows that the power saving achieved through HeVac is inversely proportional to the load, which is as expected since the idle time is inversely proportional to the load, and HeVac achieves power saving through the idle period. Although not shown in the figure, the average latency throughout the entire trace is 233µs, with standard deviation 23.2µs for HeVac. This increased latency is not significant in terms of TCP flow processing.

In Fig. 11, we show the average power saving when the thermal constraint, T_c, is varied. The x-axis is the thermal constraint percentage, which is defined as $\frac{T_{max}-T_c}{T_{max}-T_{amb}}$. We can see that HeVac can achieve 10% more power saving than VAC when thermal constraint is high (<25%), because HeVac will migrate the load and create opportunities for deep C-states. However, the advantage of HeVac is diminished when the constraint is tight (>30%), since there is no

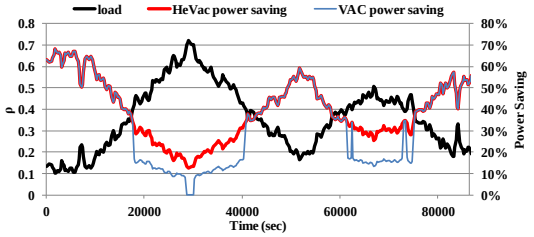

Figure 10. VAC and HeVac power saving under thermal constraint.

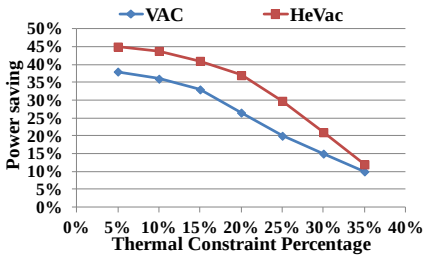

Figure 11. Trade-off between temperature and power saving.

room left for load migration without violating thermal constraint.

VI. Related Work

Researchers from the network community demonstrated the energy efficiency of per-core dynamic voltage and frequency scaling (DVFS) and clock/power gating in CMPs. In [7], Kuang et al. use DVFS for scheduling of networking applications. This scheme allocates frequencies to the pipeline stages statically and is not aimed at exploiting different traffic variations. In [17], the authors developed an analytical model to determine how many cores can be turned off, and assign different frequency to the active cores. In [18], Iqbal et al. use the traffic prediction, to determine the number of active core, and increase or decrease the frequency of the active cores. Although Per-core DVFS and clock/power gating is proven to be helpful for the temperature and power consumption, providing such a fine-grained control in CMPs with more than few cores is uncommon [19] and most of modern CMPs only support chip-wide DVFS. The system-wide performance degradation of chip-wide DVFS is its main drawback. For example, to maintain the temperature of an overheating core, the frequency of the entire chip need to be decreased. As a result, the performance of non-overheated core is compromised.

In [15,16], the authors proposed to put the network devices into power saving mode during the idle period, and wake up the processor when a packet arrives. However, this approach highly depends on the traffic pattern and the transition overhead of the power saving mode, it only achieves good energy efficiency when the idle period is larger than transition overhead. Recent studies address this issue by throttle the execution. In [20], the authors proposed "buffer and burst" for the network router. It buffers the incoming packets for a period of time, during which the network devices will stay idle. With this approach, the incoming traffic is reshaped into small bursts of arrival; devices wake up to transmit a burst of packets, and then sleep until the next burst arrives. The intent is to provide sufficient time for devices to enter the power saving mode with lower power consumption. In [20,21], the authors show the experimental results for the latency, power consumption and temperature under the effect of deferred execution. However, how to balance the trade-off between these three major performance metrics and the effectiveness of this approach in the packet processing on multicore architecture remain unclear.

In this paper, we follow similar techniques, but introduce variations by limiting our study to packet processing only and adopting existing power saving mode (C-states) on CPU. In addition, we consider the thermal behavior, which is very important in network system design [11].

References

[1] M. Dobrescu, N. Egi, K. Argyraki, et al. RouteBricks: Exploiting Parallelism to Scale Software Routers. SOSP,2009.

[2] Y. Ma, S. Banerjee, et. al. Leveraging Parallelismfor Multi-dimensional Packet Classification on Software Routers. SIGMETRICS, 2010.

[3] T. Nelms and M. Ahamad. Packet scheduling for deep packet inspection on multi-core architectures. ANCS , 2010

[4] D. Guo, L. Bhuyan, and B. Liu. An Efficient Parallelized L7-Filter Design for Multicore Servers. IEEE/ACM Transactions on Networking, October 2012.

[5] K. Jang, S. Han, et. Al. SSLShader: Cheap SSL Acceleration with Commodity Processors. NSDI, 2011.

[6] N. Tian, and Z.G. Zhang, Vacation queueing models: theory and applications . Springer, 2006.

[7] J. Kuang and L. Bhuyan. Optimizing throughput and latency under given power budget for network packet processing. INFOCOM, 2010.

[8] N. El-Sayed, I. A. Stefanovici, et. al. Temperature Management in Data Centers: Why Some (Might) Like It Hot. SIGMETRICS, 2012.

[9] J. Srinivasan, S. V. Adve, et. al. The Case for Lifetime Reliability-Aware Microprocessors. ISCA, 2004.

[10] K. Skadron, T. Abdelzaher. Control-theoretic techniques and thermal- RC modeling for accurate and localized dynamic thermal management. HPCA, 2002.

[11] J. Kuang, L. Bhuyan. Thermal-aware Scheduling of Network Applications on Multicore Architecture. ANCS, 2009.

[12] T. Benson, A. Akella, and D. Maltz. Network traffic characteristics of data centers in the wild. IMC, 2010.

[13] M. Eisenberg and K.K. Leung, A single-server queue with vacations and non-gated time-limited service, Performance Evaluation, 1991.

[14] "CAIDA Equinix", http://www.caida.org/

[15] R. Kravetsa, and P. Krishnanb. Application-driven power management for mobile communication. Wireless Networks, 2000.

[16] D. Meisner, B. Gold, and T. Wenisch. PowerNap: Eliminating Server Idle Power. ASPLOS, 2009.

[17] Jilong Kuang, L. Bhuyan. "Traffic-aware power optimization for network applications on multicore servers," DAC. 2012

[18] M.F. Iqbal, and L.K. John. Efficient traffic aware power management in multicore communications procesors. *ANCS*, 2012.

[19] W. Kim, M. S. Gupta, et. al. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. HPCA, 2008.

[20] S. Nedevschi, L. Popa, G. Iannaccone, et. al. Reducing Network Energy Consumption via Sleeping and Rate-Adaptation. NSDI, 2008.

[21] Y. Luo, J. Yu, J. Yang, and L. Bhuyan. Conserving network processor power consumption by exploiting traffic variability. TACO, 2007.

[22] J. Koomey. Estimating total power consumption by servers in the us and the world. Analytics Press, 2007.