Octopus-Man: QoS-Driven Task Management for Heterogeneous Multicores in Warehouse-Scale Computers

Vinicius Petrucci*[1], Michael A. Laurenzano†, John Doherty†, Yunqi Zhang†, Daniel Mossé‡, Jason Mars†, Lingjia Tang†

Clarity Lab
†University of Michigan, Ann Arbor, MI, USA
{*mlaurenz,johndoh,yunqi,profmars,lingjia*}@*umich.edu*

*Federal University of Bahia, Salvador, BA, Brazil
petrucci@dcc.ufba.br

‡University of Pittsburgh, Pittsburgh, PA, USA
mosse@cs.pitt.edu

***Abstract*— Heterogeneous multicore architectures have the potential to improve energy efficiency by integrating power-efficient wimpy cores with high-performing brawny cores. However, it is an open question as how to deliver energy reduction while ensuring the quality of service (QoS) of latency-sensitive web-services running on such heterogeneous multicores in warehouse-scale computers (WSCs).**

In this work, we first investigate the implications of heterogeneous multicores in WSCs and show that directly adopting heterogeneous multicores without re-designing the software stack to provide QoS management leads to significant QoS violations. We then present *Octopus-Man*, a novel QoS-aware task management solution that dynamically maps latency-sensitive tasks to the least power-hungry processing resources that are sufficient to meet the QoS requirements. Using carefully-designed feedback-control mechanisms, Octopus-Man addresses critical challenges that emerge due to uncertainties in workload fluctuations and adaptation dynamics in a real system. Our evaluation using web-search and memcached running on a real-system Intel heterogeneous prototype demonstrates that Octopus-Man improves energy efficiency by up to 41% (CPU power) and up to 15% (system power) over an all-brawny WSC design while adhering to specified QoS targets.

I. Introduction

The design methodology of modern *warehouse-scale computers* (WSCs) [5] has been architecturally-homogeneous designs [3], [6] that often exclusively use *brawny* (complex, out-of-order) core types. As noted by recent works [17], [42], commodity server grade chips such as Intel Xeons and AMD Opterons comprise the entire fleet of Google's and the majority of Facebook's WSC infrastructures as they are cheap and easily replaceable. These brawny processors deliver high performance, but at the cost of high power consumption.

While *wimpy* cores (simple, in-order architectures) offer lower performance, they have significantly higher power efficiency. There is a significant amount of prior work advocating *wimpy* cores for WSC design in both industry [7], [21] and academia [2], [26], [27], [38]. Although the energy efficiency of commodity wimpy cores can be realized for batch and throughput-oriented workloads hosted in WSCs, the resulting performance and quality of service (QoS) degradation of latency-sensitive applications prohibits adopting wimpy cores in production [24]. These latency-sensitive applications require tight constraints on the QoS because they interact directly with users. In particular, the tail of the latency distribution must be kept below a QoS threshold [9]. Although wimpy cores can provide advantages in power, cost, and performance per dollar, web-service companies still primarily deploy and rely on brawny cores to deliver high single-threaded performance for complex latency-sensitive applications [24], [49].

Another architectural design point integrating both wimpy and brawny cores may be promising for future server designs. Examples of these *heterogeneous multicore architectures* include ARM's big.LITTLE [21], for mobile platforms, and Intel's QuickIA prototype [7], for server platforms. In these designs, wimpy and brawny cores share the memory address space and are visible to a single operating system, enabling fast task migration among the cores on a server. Prior work on managing heterogeneous core resources is driven by either hardware performance counters (e.g., instructions/cycle, cache misses) [52] or CPU utilization [58]. Therefore, although prior work has demonstrated the potential of heterogeneous multicore systems, the proposed systems do not provide explicit QoS management or QoS guarantee for latency-sensitive services in WSCs.

In this work, we first investigate the implications of leveraging heterogeneous multicore architectures for WSCs using a real system prototype from Intel, the QuickIA platform [7], and WSC workloads including web-search and memcached. We find that simply replacing the underlying hardware for heterogeneity without redesigning the system software stack results in an unacceptable increase in QoS violations. In addition, load fluctuations in WSCs, such as diurnal load changes, present opportunities for energy efficiency that can be exploited by migrating tasks between the heterogeneous cores based on the dynamic load. Therefore, an intelligent QoS-aware runtime system that manages brawny and wimpy core resources to deliver energy efficiency gain while guaranteeing QoS is critical to a WSC composed of heterogeneous multicores. Such a system must address two major design challenges:

[1] Work was conducted as a postdoc fellow of Clarity Lab at the University of Michigan.

- *Responsiveness* - The QoS-aware runtime system needs to readily respond to changes in the execution environment such as time-varying load fluctuations and spikes, and promptly adapt the system to meet the QoS targets in the presence of these changes.
- *Stability* - The QoS-aware runtime needs to prevent oscillatory behavior that unnecessarily and frequently switches between system configurations such as core mappings for the latency-sensitive applications. Such oscillations can negatively affect the application QoS.

In this paper, we design and prototype **Octopus-Man**, a QoS-aware task management system that addresses these challenges and dynamically manages tasks on heterogeneous multicores. Driven by runtime QoS measurements, Octopus-Man exploits load changes to allocate latency-sensitive tasks to the least power-hungry processing resources that are sufficient to meet the QoS requirements. By continuously monitoring QoS changes and carefully expanding, contracting and migrating between wimpy/brawny core resources, Octopus-Man is able to meet tail latency requirements, while minimizing power consumption and improving server energy efficiency. Octopus-Man requires no extra hardware support beyond the heterogeneous multicore substrate and no modification to the host OS.

This work makes the following specific contributions:

- **Investigation of Heterogeneous Multicore in WSCs —** We perform an investigation and describe the opportunities and challenges of using heterogeneous multicore servers for improving energy efficiency in WSCs. We find that runtime task management is critical for meeting QoS targets in latency-sensitive web services. (Section III)
- **Octopus-Man Task Management —** We present Octopus-Man, an adaptive runtime system for managing task assignment to heterogeneous core resources in WSCs while reducing power consumption and ensuring that QoS targets are met. (Section IV)
- **Real System Prototype and Evaluation —** We design and deploy a functional prototype Octopus-Man along with web-services including web-search (Apache Nutch) and data-caching (memcached). The evaluation uses real production load intensity traces from Google and real-system prototype Intel's QuickIA [7]. (Section V)

Our experimental results show that Octopus-Man reduces CPU energy consumption by up to 41% and server-level energy by up to 15%, while meeting the QoS targets on tail latency as effectively as all-brawny WSC designs. We also show that Octopus-Man can improve batch processing throughput by 34% over the current all-brawny systems.

II. Background

In this section, we first introduce the server selection approaches, job scheduling techniques and quality of service metrics that are commonly used in modern *warehouse-scale computers* (WSCs). We then present an overview and discussion on heterogeneous multicore architectures for WSCs.

A. Warehouse Scale Computers

Job Characteristics — WSCs host a fleet of machines grouped into one or more clusters. Users submit jobs to the cluster of machines, where a job consists of one of more tasks [47]. There are two classes of jobs in WSCs: latency-sensitive *service jobs* and throughput-oriented *batch jobs*. Latency-sensitive jobs must meet certain QoS requirements and often experience a time-varying pattern in their load. Batch jobs, on the other hand, are typically throughput oriented and do not have strict QoS constraints.

Cluster Scheduling — A cluster-wide scheduler is responsible for placing and tracking the jobs on the available platform resources [47], [53]. Each job's required resources, such as the number of processing cores and amount of memory needed, is specified in a configuration file associated with the job. Given the resource requirements, the cluster scheduler uses a variant of the bin-packing algorithm to make job mapping decisions [47], [53].

Quality of Service — QoS for service jobs in WSCs is typically defined in the form of service level objectives (SLOs) using statistical guarantees of application-level metrics such as query latency. In addition to the average or median query latency, quality of service often focuses on the tail distribution of the latency to improve interactivity [9]. For example, a quality of service target could be specified as "90% of the search queries need to have a latency under 500 ms."

B. Heterogeneous Multicore Architectures

Brawny vs. Wimpy — Although wimpy cores can be more power efficient, most WSCs currently adopt homogeneous brawny servers to ensure the quality of service for latency-sensitive service jobs. This is because there are several limitations in using wimpy cores that affect their broad adoption in WSCs [24]. In some cases, single-threaded performance is still important due to Amdahls' law; the inherently serial computation by slow wimpy cores can dominate overall execution time. Using wimpy cores requires additional parallelization efforts to deal with many more subtasks. The variability in the tail of latency distribution is also amplified with increased number of computing units in the data center, since many more wimpy cores would be needed to sustain the same performance as that of brawny cores. This makes it more difficult to deliver satisfactory latency requirements.

Heterogeneous multicores — Emerging heterogeneous multicore designs [30] exploit the fact that applications' resource requirements are different. Thus, by combining cores that trade off performance with power to different degrees, the resulting system can be more energy efficient than homogeneous systems. Several such architectures have come to fruition, including ARM's big.LITTLE [21] and Intel's QuickIA prototype [7]. QuickIA is a prototype for server-class computing platforms that integrates Intel's Atom and Xeon cores. Given that our focus is on WSCs, in this paper we focus our efforts on QuickIA (Section V).

III. Heterogeneous Multicores in WSCs: Opportunities and Challenges

In this section, we investigate the performance and energy efficiency trade-offs for wimpy cores (Intel Atom) and brawny cores (Intel Xeon) running a typical WSC workload (web-search). We present the opportunities and challenges of using

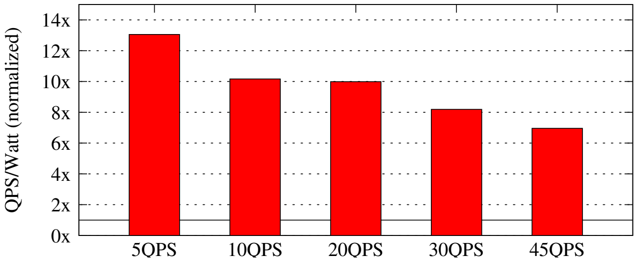


Fig. 1: Throughput per watt of web-search on wimpy cores at various load levels (normalized to brawny cores)

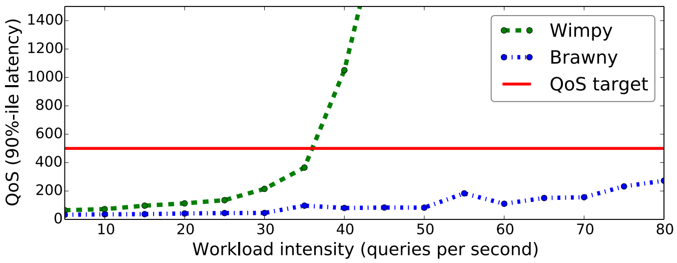


Fig. 2: 90th percentile query latency of web-search on wimpy vs. brawny cores at various load levels

heterogeneous multicore servers for improving energy efficiency in WSCs, highlighting the need for an intelligent QoS-aware runtime system to achieve both satisfactory performance and high energy efficiency.

A. Wimpy vs. Brawny for WSC Workloads

We first study the energy efficiency and performance trade-offs between wimpy and brawny cores using Intel's QuickIA heterogeneous platform [7] and the web-search (Apache Nutch) benchmark from CloudSuite [16].

Energy Efficiency — Figure 1 presents the energy efficiency in $QPS/Watt$ (queries per second per Watt) of a wimpy core running a web-search job at different load levels, normalized to running on a brawny core. In this experiment we account for the power consumption of each processor type in isolation. More details about the experimental setup are presented in Section V. Figure 1 demonstrates that wimpy cores are much more energy-efficient than brawny cores for web-search jobs at various loads. Note that, as the load increases, the energy-efficient benefit of wimpy cores decreases. In fact, wimpy cores are much more energy-efficient at low load because wimpy cores have very low idle power and larger difference between the idle and peak power (better energy proportionality) than brawny cores [49].

Query Latency — Figure 2 presents the 90^{th} percentile query latency achieved by wimpy cores vs. brawny cores at different load levels for web-search. As shown in the figure, when the load is low, especially below 20 QPS, the 90^{th} percentile query latency for wimpy and brawny cores are relatively similar, as the web-search job does not put a lot of pressure on the system. However, as the load intensity increases, the query latency on wimpy cores increases rapidly, and the gap in latency between wimpy and brawny cores becomes much larger. At 40 QPS, the 90^{th} percentile latency on wimpy cores is 13X longer than the 90^{th} percentile latency on brawny cores. Considering the 90^{th} percentile query latency at 500ms as the QoS target, a wimpy core can sustain up to 35 QPS, while a brawny core can support as many as 80 QPS.

This figure shows that while brawny cores clearly achieve lower latency at all load levels, at low load the latency delivered by the wimpy core can be adequate and comes at a much lower power cost, making wimpy cores an attractive alternative at low load. The need for both high performance cores at high load and energy efficient cores at low load lends itself to core heterogeneous designs, which offer both core types in a single server.

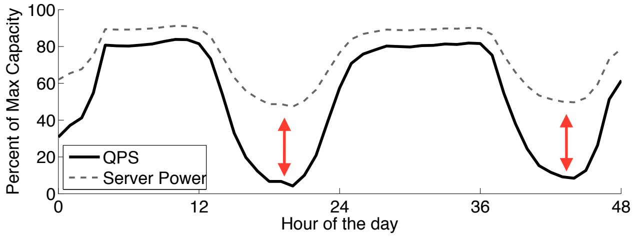


Fig. 3: Load fluctuation and power consumption for web-search running on Google servers, adapted from Meisner et al. [46]

B. Energy Reduction Potentials of Heterogeneous Platforms

Figure 3, adapted from Meisner et al. [46], illustrates the typical load intensity (in queries per second) and power consumption for a web-search service running on Google servers. This class of workload presents a wide dynamic load range, experiencing long periods of low load. Moreover, during the periods of low load, the power consumption of (current) brawny servers is still relatively high; that is, the energy consumption is not proportional to the amount of computation accomplished by the brawny server [4].

Our insight is that the load varying nature of service jobs coupled with the fact that, at low loads, the performance of wimpy is sufficiently close to that of brawny, rendering the core heterogeneous design point an attractive option. Heterogeneous multicore architectures present the opportunity to combine the best of both worlds by allowing tasks to quickly migrate between wimpy and brawny cores at runtime to meet the QoS target of service jobs and achieve high energy efficiency.

C. Challenges of Adopting Heterogeneous Platforms

Despite the potential for energy optimization when employing core heterogeneous designs, QoS challenges arise when directly adopting heterogeneous platforms in WSCs without an effective task management runtime. To understand the need to introduce such a QoS-aware runtime, we perform an analysis of simply replacing current all-brawny cores with heterogeneous multicores without redesigning the system software stack.

Figure 4 presents the number of QoS violations given various latency targets for web-search queries. We compare a WSC composed of homogeneous servers with 4 brawny cores (Xeon) per server versus a WSC composed of heterogeneous

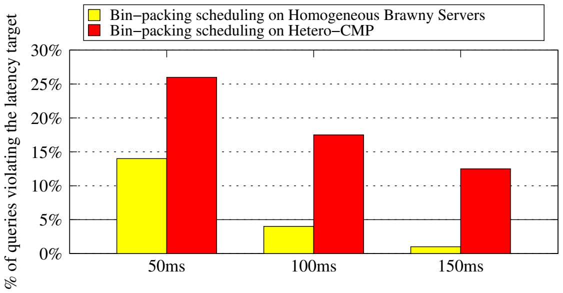


Fig. 4: QoS violations for web-search for homogeneous brawny servers vs. heterogeneous servers at various latency targets.

servers with 2 brawny and 2 wimpy (Atom) cores per server. In this experiment, each web-search task requires 2 cores and is mapped to each server by the cluster scheduler using the standard bin-packing algorithm commonly found in modern WSCs [47], [53]. The bin-packing algorithm randomly assigns each job to a machine that has at least as many available cores as are needed by the job.

As shown in Figure 4, using a standard bin-packing algorithm that is unaware of the heterogeneity or QoS causes significantly more QoS violations with heterogeneous platforms. For example, 17% of all queries on the heterogeneous platform violate their QoS target of 100ms. For heterogeneous multicore to be a viable option in WSC design, we must provide a mechanism to allow latency-sensitive jobs to meet QoS targets while optimizing for energy efficiency. This is the primary goal of Octopus-Man.

IV. Octopus-Man

In this section, we introduce **Octopus-Man**, a runtime system that manages brawny/wimpy core resources in WSCs, adaptively allocating the necessary and the least power-hungry core resources for latency-sensitive applications to ensure QoS while maximizing energy efficiency.

A. Design Goals and Challenges

The major design goal of Octopus-Man is to ensure that the QoS targets of latency-sensitive jobs are satisfied and to maximize the energy efficiency. Some WSCs co-locate latency-sensitive applications (service jobs) and batch applications (throughput-oriented jobs) on shared servers to improve server utilization [42]. For the shared servers, Octopus-Man ensures that the QoS target of latency-sensitive jobs is met while maximizing the throughput of batch jobs.

Octopus-Man is designed to address the important challenge of achieving both *responsiveness* and *stability*. Firstly, Octopus-Man must be able to detect and be responsive to changes in the system's execution environment such as workload fluctuations and co-location interferences. It must control the system via migrating tasks to appropriate core resources to ensure that the managed applications meet QoS goals in the presence of these changes. Secondly, Octopus-Man must prevent the server system from unnecessary oscillations between core mappings, negatively affecting the QoS.

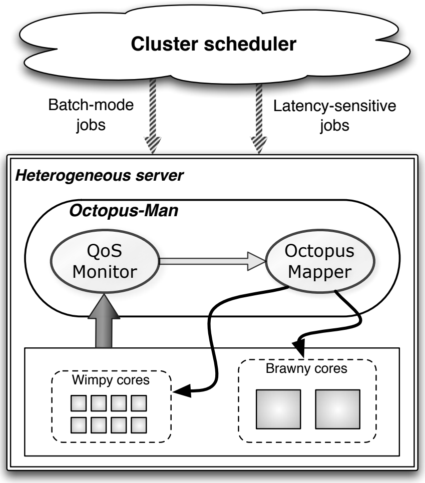


Fig. 5: Overview of Octopus-Man runtime system

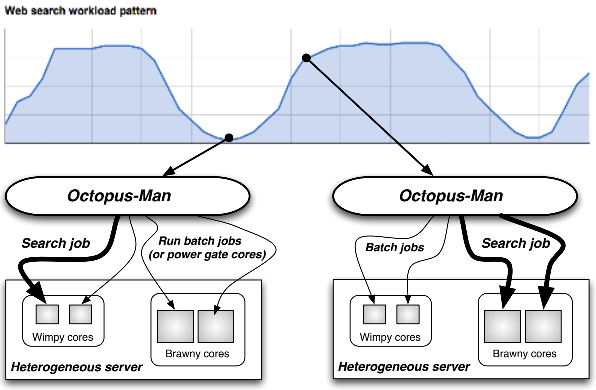
B. Octopus-Man Overview

Octopus-Man is built upon two insights: 1) heterogeneous multicores allow tasks within a server to be quickly migrated between wimpy and brawny cores at runtime, and 2) the latency difference between brawny and wimpy cores is significantly smaller at low load. By leveraging load-fluctuations, Octopus-Man migrates latency-sensitive applications to wimpy cores during periods of low load to achieve high energy efficiency without impacting user experience.

The *Octopus-Man* framework is depicted in Figure 5. Octopus-Man consists of two main components: the *QoS Monitor* and the *Octopus Mapper*.

QoS Monitor — The QoS monitor is responsible for collecting job performance data from the heterogeneous server using in-place continuous profiling. Existing runtime monitoring systems are deployed in modern WSCs to continuously gather detailed task performance information [50]. The Octopus-Man system is designed to leverage these light-weight monitoring systems. There are a number of performance and QoS metrics available to Octopus-Man including the application-level metrics (e.g., query latency for web-search) and operating system/hardware performance counter metrics (e.g., CPU utilization, instructions per cycle and cache misses).

Octopus Mapper — Based on dynamic performance profiles collected by the QoS Monitor, the Octopus Mapper makes job mapping decisions and adapts the heterogeneous server system to improve energy efficiency. Figure 6 illustrates an execution example of the Octopus-Man when managing co-located batch jobs and a web-search (service job) on a shared server following the fluctuating load of a typical diurnal pattern. As shown in Figure 6-A, during the periods of low utilization of the service job, the Octopus-Man system may run web services on wimpy cores while turning off some power-hungry brawny cores to minimize energy consumption (on a dedicated server for latency sensitive applications) or using the available brawny cores to accelerate batch processing (on a shared server hosting both batch and latency-sensitive jobs). When the service job experiences high load intensity, as seen



(A) Contracting Octopus *(B) Expanding Octopus*

Fig. 6: Octopus-Man managing a web-search and batch jobs, (A) contracting during low intensity load and (B) expanding during high intensity load

in Figure 6-B, the Octopus-Man system allocates the high-performance brawny cores to the latency-sensitive service job to meet its specified QoS target.

In the rest of this section, we will describe in details the design of Octopus-Man Mapper including its algorithms to determine the appropriate task migration strategies. We will first introduce the formal notations for core mappings and transitions in Section IV-C. We will then present two designs of the decision algorithms of Octopus-Man Mapper in Sections IV-D and IV-E. Finally we will present how Octopus-Man addresses two critical design challenges and the implementation details in Sections IV-F and IV-G.

C. Core Mapping and Transition

A heterogeneous multi-core system has a discrete set of core configurations. We consider N wimpy cores and M brawny cores in a system, where the wimpy core set is $W = \{w_1, \ldots, w_i, \ldots, w_N\}$ and the brawny core set is $B = \{b_1, \ldots, b_j, \ldots, b_M\}$. We then specify a resource allocation set A, in ascending order based on the power consumption of each core configuration as follows:

$A = \{\{w_1\}, \{w_1, w_2\}, \ldots, \{w_1, w_2, \ldots, w_N\}, \{b_1, \ldots, b_L\},$
$\{b_1, \ldots, b_L, b_{L+1}\}, \{b_1, \ldots, b_L, b_{L+1}, \ldots, b_M\}\}$

When migrating a latency-sensitive job from wimpy to brawny cores, we define L as the minimal number of brawny cores necessary to provide the same processing capacity as N wimpy cores. This ensures that the new job mapping does not violate the service QoS target after the migration.

Octopus-Man determines the core mapping based on QoS, for example, by increasing the core resources when QoS is low. This methodology is general and can handle mappings that include both wimpy and brawny cores. In our system we do not consider a service task running simultaneously on both wimpy and brawny cores because there is no noticeable QoS improvement on our heterogeneous platform, Intel QuickIA, for this mixed mapping over brawny/wimpy-only mappings. In addition, allocation of CPU resources in modern data centers is typically at the granularity of whole cores whereby each task has a reservation of cores and do not timeshare with other applications [11]. Therefore, we also do not consider service and batch jobs sharing the same core.

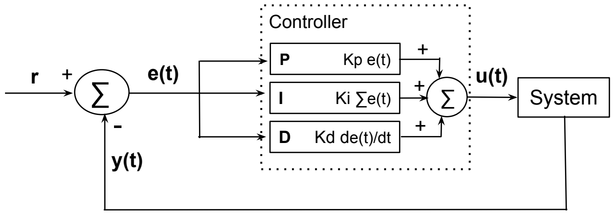


Fig. 7: Basic elements of the PID control system

Next, we present two designs for the mapping algorithm used in the Octopus-Man mapper: (1) a Proportional-Integral-Derivative (PID) control system [23] and (2) a deadzone state machine. PID controller is a classic control loop feedback algorithm widely used in industrial control systems [23]. Our deadzone state machine is inspired by a real-time deadzone-based scheme for voltage scaling and latency control in web servers [25].

D. Design 1: PID Control System

We first investigate using PID control in Octopus-Man to decide the appropriate task mapping and migration on a heterogeneous multicore server. The goal of PID control is to make sure that the underlying controlled system operates as close as possible to a specified set point. In our case, PID control aims to manage the underlying system, i.e., latency-sensitive tasks executing on a heterogeneous multicore, so that the achieved QoS of the system is as close as possible to the specified QoS target. To this end, the PID controller continuously monitors the system feedback, i.e., QoS of the system, and adapts the system configurations, i.e., the task mappings on the heterogeneous multicore accordingly over time.

Figure 7 presents the basic structure of a PID controlled feedback loop. As depicted in Figure 7, the PID control takes a controller reference r as the desired QoS target (e.g., 90%-ile latency at 500ms). The system feedback is given by the controller error signal $e(t)$, defined as the difference between the QoS target r and the current monitored QoS value $y(t)$ at time t; that is, $e(t) = r - y(t)$. Based on the control error $e(t)$, the PID control system calculates three terms: the proportional (current error rate), the integral (overall net errors) and the derivative (correction rate) of the error signal. The PID controller equation $u(t)$ then associates each term with a weight (K_p, K_i and K_d) and sums up the three terms as follows:

$$u(t) = K_p e(t) + K_i \sum_{k=0}^{t} e(k)T + K_d \frac{e(t) - e(t-1)}{T} \tag{1}$$

The controller output $u(t)$ from Equation 1 indicates the appropriate system configuration for minimizing the error in the next sampling interval error, $e(t+1)$. In the case of Octopus-Man, $u(t)$ directly dicates the appropriate wimpy/brawny core configuration the system should be adjusted to to achieve the QoS target.

Since the PID control output $u(t)$ has a continuous range and our heterogeneous system has a discrete set of core configurations, we use a mapping scheme that relates the control output to the set of core configurations. We set the minimum and maximum values of the controller output (e.g., $min\_u = 0$ and $max\_u = 255$), and any value between the controller limits is linearly scaled to a position in the set of wimpy/brawny core configurations (set A specified in Section IV-C). The min_u is mapped to one wimpy core $\{w_1\}$ and max_u is mapped to $\{b_1, \ldots, b_L, b_{L+1}, \ldots, b_M\}\}$ brawny cores.

The controller's sampling interval T specifies how often the monitored QoS variables are sampled at runtime. The choice of an appropriate sampling interval should be based on the dynamics of the system being controlled, and Section IV-F describes our methodology for deriving this parameter.

PID Controller Configuration — To carefully tune and configure PID to achieve high-quality control, that is to determine appropriate values for parameters K_p, K_i and K_d in Equation 1, we employ the commonly used root locus method [18].

To use root locus method to determine the suitable parameter values, we need to first identify a transfer function of our system. The transfer function is a formal representation of the input-output relationship for the system (i.e., input as measured QoS of the system and output as core mappings). Root locus method then conducts sensitivity analysis of the parameters based on the transfer function to select suitable parameter values for a high-quality, stable controller.

As in prior work [39], we use profiling data to build a model to determine the transfer function of our system. We collect the average latency of queries for each application at different load levels and build a linear regression model that predicts the QoS for a given system configuration (core mapping) as shown in Equation 2. The system input (u) represents the core configurations and (y) represents the corresponding system output, i.e., the QoS of the system; where m represents the time at which data is collected from the system and n is the order of the equation modeling the system. We use $n = 1$ to build our model that best represents our system. We model the system using the differential equation [19] and use the least squares [39] to determine the g and h parameters in Equation 2.

$$y(m) = \sum_{i=1}^{n} g_i u(m-i) + \sum_{i=1}^{n} h_i y(m-i) \tag{2}$$

Transforming Equation 2 into the discrete time domain, we arrive at Equation 3. Combining Equation 3 and Equation 1, we arrive at Equation 4, a combined representation of the controller parameters and the controlled system in discrete time domain.

$$P(z) = \frac{y(z)}{u(z)} = \frac{h_1}{z - g_1} \tag{3}$$

$$C(z) = \frac{u(z)}{e(z)} = K_p + K_i \frac{T}{z-1} + K_d \left(\frac{z-1}{T}\right)^2 \tag{4}$$

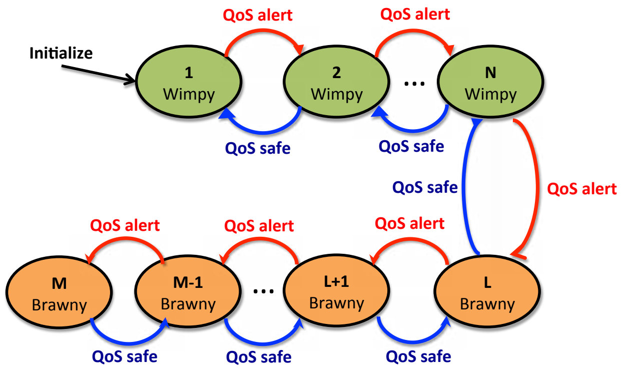


Fig. 8: State machine used in deadzone-based Octopus Mapper

$$G(z) = \frac{C(z)P(z)}{1 + C(z)P(z)} \tag{5}$$

We then perform a root locus analysis using Equation 5 [18] to narrow down an appropriate range of parameters (K_p, K_i and K_d) for the PID controller for each latency-sensitive application. Using root locus, we evaluate the behavior of the PID controller when the gain K_p is adjusted; K_i and K_d are tuned based on the values of the gain. The method helps visualize the stability of the controller and observe the effects of varying the gain. We then performed a parameter tuning over that range to determine the parameters to achieve a stable controller.

E. Design 2: Deadzone State Machine

Here we present a deadzone based design of Octopus-Man mapper, which models the Octopus-Man's task mapping algorithm using a state machine. The Octopus-Man state machine, illustrated in Figure 8, consists of a set of states and transitions between those states. The available core configurations in the controlled system are represented by the possible states. Note that each element from the set of wimpy/brawny core configurations A (Section IV-C) is mapped to a state in the transition system. At any given time, the Octopus-Man state machine is in only one state: the current state. The transition from one state to another is initiated by triggering conditions specified by QoS rules. Octopus-Man uses this representation to map the latency-sensitive job to N wimpy cores or M brawny cores to optimize for energy reduction or job throughput while respecting latency constraints.

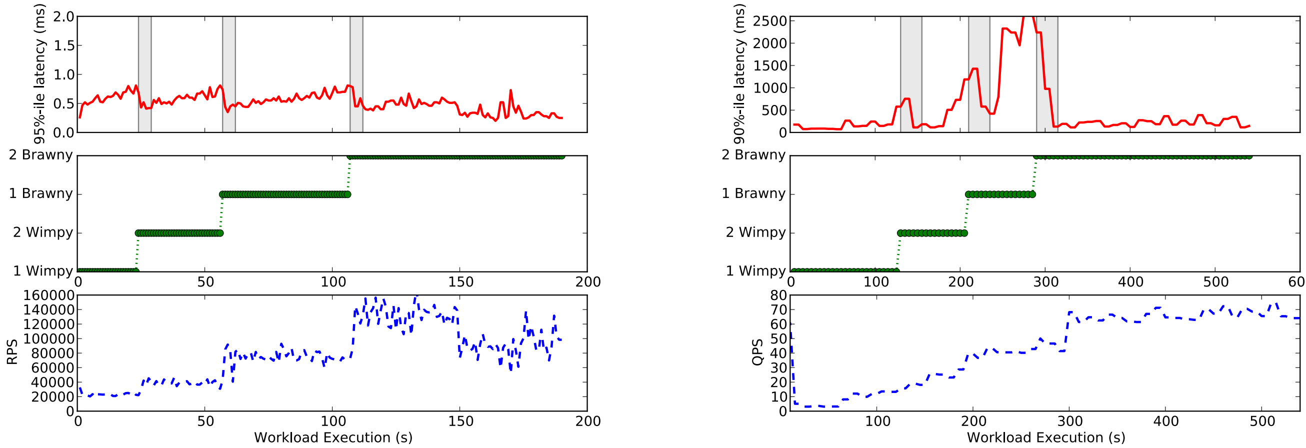
For each latency-sensitive job controlled by Octopus-Man, we specify a *QoS controlled variable* as the percentile (e.g, 90^{th} or 95^{th}) of the monitored request latency and a *QoS target* that the system needs to ensure. Then the **QoS alert** trigger condition is defined as:

$$QoS_{variable} > QoS_{target} \cdot UP_{thr}$$

The **QoS safe** trigger is defined as:

$$QoS_{variable} < QoS_{target} \cdot DOWN_{thr}$$

The QoS rules state that when the measured QoS latency variable exceeds an *upper bound* (or drops below a *lower*



(A) Memcached workload

(B) Web-search workload

Fig. 9: Detecting adequate settling times (gray area in the figures) due to core switching

bound), Octopus-Man expands (or contracts) the number of cores allocated to the latency-sensitive job to meet its specified QoS target. The rationale for the upper threshold is that, for any two consecutive QoS measures from the workload distribution, the following conditional probability as a function of the UP_{thr} must be satisfied [25]:

$$P(QoS_{k+1} > QoS_{target} | QoS_k < UP_{thr} \cdot QoS_{target}) <= QoS_{vio}$$

The QoS_{vio} is the maximum percentage of QoS violation expected in the system (e.g., 5%). The upper threshold UP_{thr} is obtained empirically from the workload distribution as in [25]. An adequate down threshold $DOWN_{thr}$ is then selected to minimize oscillation while still delivering energy efficiency. A *sampling interval* is used to periodically evaluate the QoS trigger rules and potentially perform the state transitions in the system. The details on selecting these parameters are discussed in Section IV-F.

F. System Responsiveness and Stability

Here we describe how Octopus-Man addresses the following two important challenges: *Responsiveness* and *Stability*.

Responsiveness — To meet the responsiveness criterion, we need to determine the suitable *sampling interval*, which specifies how often Octopus-Man should sample the monitored QoS and decide whether to transition to a new task-to-core mapping configuration.

We observe that typical WSC applications (e.g., web-search and memcached) often exhibit short-term high-variability QoS when subjected to excessively frequent dynamic core switching. The sampling interval thus should consider the time required for (A) performing task-to-core dynamic configuration and (B) waiting for the updated QoS measurements to stabilize; that is, reaching and remaining within a relatively small range (e.g., 5%-10%) of the final stable QoS measure. This waiting time is known as *settling time* in control systems [18].

We specify the sampling interval as the minimum monitoring interval provided by each latency-sensitive application plus the required settling time. This allows Octopus-Man to react quickly to fluctuating load without causing excessive switching behaviors. The default monitoring (QoS reporting) interval for Memcached is every second, whereas web-search (Apache Nutch) typically requires about five seconds to update the logs of query processing times. In both cases, the overhead of switching cores on/off and migrating tasks are negligible (in the order of microseconds to few milliseconds [8], [33], [40]).

To determine an appropriate settling time for each latency-sensitive application, we perform an automatic profiling. In the profiling, we change the core mapping to observe the impact on the QoS over time. Figure 9 presents varying core configurations for each application and its impact on QoS and queries per second. The gray area in the figure represents the settling time needed for memcached and web-search application. This corresponds to the time interval from the moment of a core switching until the slope of two consecutive QoS measures stabilizes (less than 10% variation between successive measurements).

Stability — Stability is an important issue for both PID control and deadzone state machine designs. For PID control, we rely on the root locus methodology to configure a stable controller [18]. For the deadzone state machine, we notice that the QoS trigger rules allow for (1) anticipating the QoS violation by setting an upper threshold to quickly allocate resources before QoS violations even occur, and 2) minimizing oscillatory behavior by using the down threshold when deallocating resources. To achieve stability, we need to determine the appropriate values for these thresholds, especially the down threshold.

Figure 10 highlights the importance of selecting an adequate threshold parameter to avoid the oscillatory behavior. In this experiment, we notice that 46% of QoS violations are due to excessive task migrations. We propose a solution to address this issue by detecting such oscillatory behavior and adjusting the threshold parameter to reduce the oscillation's negative impact on the application QoS.

Figure 11 illustrates our mechanism for the deadzone state

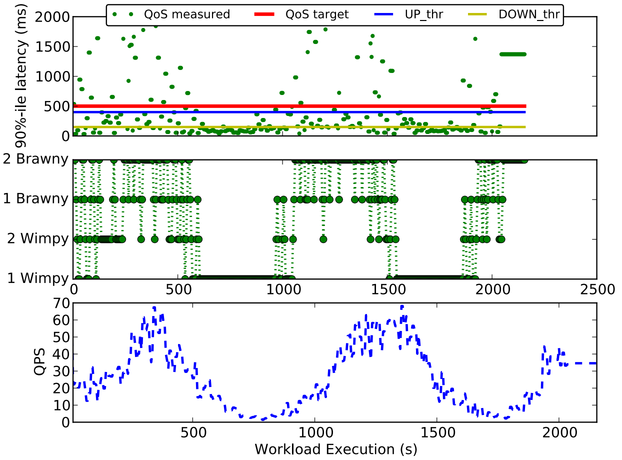


Fig. 10: Web-search execution when deadzone thresholds are set as UP_thr=0.8, DOWN_thr=0.3. High QoS violations occur due to oscillatory behavior caused by inappropriate threshold values

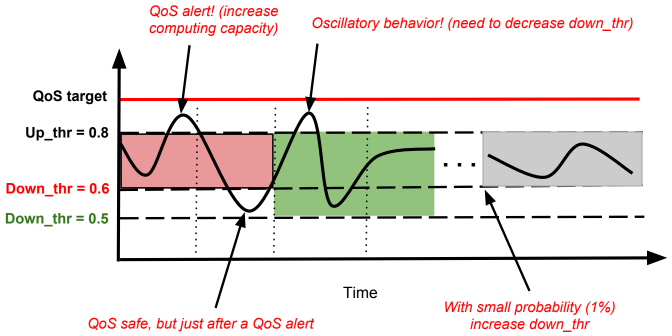

QoS alert! (increase computing capacity)
Oscillatory behavior! (need to decrease down_thr)
QoS target
Up_thr = 0.8
Down_thr = 0.6
Down_thr = 0.5
Time
QoS safe, but just after a QoS alert
With small probability (1%) increase down_thr

Fig. 11: Illustration of dynamically selecting the deadzone thresholds

machine to automatically select the down threshold parameter for a given application. The idea is to identify an oscillatory pattern characterized by a QoS_{alert} event followed immediately by a QoS_{safe} event and then followed by another QoS_{alert}. Once this oscillatory pattern is detected, Octopus-Man adjusts the down threshold to accommodate the QoS variability when adapting the system, for example, adjusting the threshold from 0.6 to 0.5 as shown in the figure. This works because increasing the service capacity (e.g., moving a service job from wimpy to brawny cores) should not make the QoS variable drop below the down threshold and trigger an unnecessary new adaptation (e.g., moving the service job back to wimpy cores) in the next sampling interval. We present in Section V experiments evaluating our adaptive deadzone technique.

G. Implementation Details

Octopus-Man is a user-level process running on Linux OS that consists of *monitor* and *mapper* modules. The monitor module collects runtime measurements of the query/request latency via a log-file (in RAM filesystem) interface with running service jobs such as the Apache Nutch web server and memcached server. The monitor module also reports the aggregated IPS (instructions per second) by measuring the hardware performance event `retired_instructions` of each core on each monitoring interval [56]. We build an instance of the monitor module for each latency-sensitive application. The actuator module is responsible for binding tasks to cores via Linux `sched_setaffinity` system call. Octopus-Man quickly suspends/resumes the batch jobs via OS signals (`SIGSTOP` and `SIGCONT` in Linux) to manage the execution of batch jobs on the heterogeneous cores.

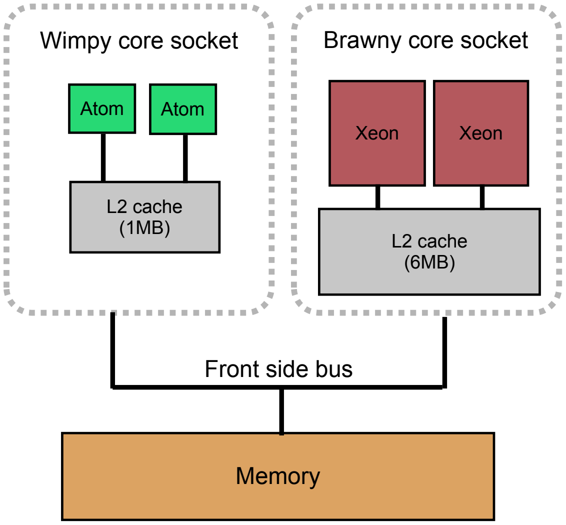


Fig. 12: Heterogeneous processor platform (Intel's QuickIA)

Core type	Peak power	Idle power
Xeon 5450	18.75 W	9.625 W
Atom N330	2.15 W	0.7 W

TABLE I: Power consumption of heterogeneous cores [26]

V. Evaluation

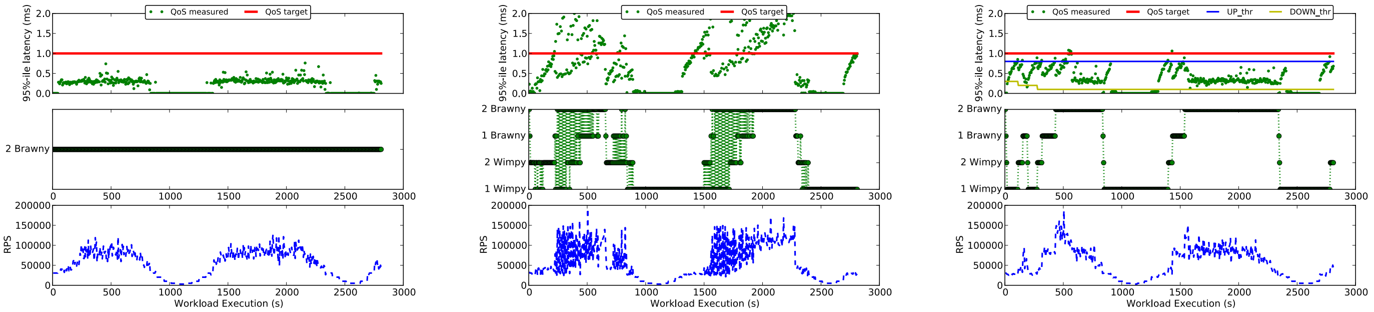
We evaluate Octopus-Man on Intel QuickIA platform using web-search (Apache Nutch) and data-caching (memcached) workloads from CloudSuite [16].

A. Intel QuickIA Prototype

We use the Intel QuickIA platform [7] which integrates a high-performance brawny processor (Xeon) and a low-power wimpy processor (Atom) on the same platform (Figure 12). This heterogeneous platform provides core types with different micro-architecture designs (simple in-order vs. aggressive out-of-order) but the same ISA.

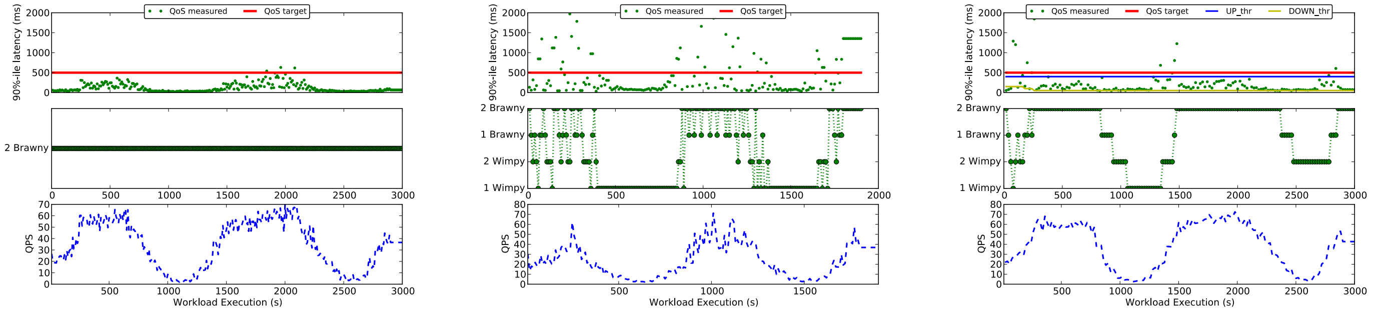
Table I summarizes the CPU power consumption for the Intel Xeon/Atom cores that comprise our Quick IA prototype [26], showing that the wimpy core has a very low idle power and also consumes on average much less (9.7× less) power than a brawny core.

We quantify *CPU power* using power measurements and CPU utilization as in prior work [14], [51], [60], where the power consumption is linearly approximated using CPU utilization. This model is used to characterize the benefits of



(A) Static (always brawny cores) *(B) Octopus-Man (PID Control)* *(C) Octopus-Man (Deadzone)*

Fig. 13: Memcached execution on QuickIA. PID performs 10× more task migrations than Deadzone, and has more QoS violations



(A) Static (always brawny cores) *(B) Octopus-Man (PID Control)* *(C) Octopus-Man (Deadzone)*

Fig. 14: Web-search execution on QuickIA. PID performs 2.5× more task migrations than Deadzone, and has more QoS violations

core heterogeneity in the presence of core-level power gating techniques found in modern CPUs.

We also provide system power measurement directly using a *WattsUp Pro* power meter [12]. These measurements reflect the power consumed by the entire machine (the power drawn at the outlet), including the CPUs. We report the dynamic power consumption by subtracting the power consumed when the machine is completely idle.

B. Workload Configurations

We adapted the web-search workload generator (Faban) and the memcached client generator (from Cloudsuite) to issue query requests driven by a time-varying production load trace using the workload profile as shown Figure 3 and described in previous work [46]. We specify the target QoS for web-search as 500ms (90^{th} percentile query latency) as prescribed by CloudSuite [16]. For memcached, we use 1ms (95^{th} percentile query latency) as the QoS target. This was determined by measuring the latency achieved by the brawny cores on our experimental platform at peak load (80% of the maximum possible load). Note that this QoS target reflects the capability of our experimental machine, and is actually much lower than the QoS target (10ms) specified in Cloudsuite [16]. The maximum-capacity load issued by the load generator is configured so that web-search and memcached running on brawny cores can meet the specified query latency target.

C. QoS Guarantee and Energy Reduction

We evaluate the effectiveness of Octopus-Man on meeting the QoS requirements of web-search and memcached, while exploiting load fluctuations and mapping the tasks to the most suitable heterogeneous cores in the system to achieve high energy efficiency.

Memcached — Figure 13-A shows the performance of the memcached workload using the baseline **Static** all-brawny core mapping during a 48-hour period [46]; each hour in the original workload corresponds to one minute execution in our experiments. The top plot presents the 95^{th} percentile QoS behavior; the target is 1 ms indicated by the red line. The middle plot presents mapping decisions and the bottom plot shows the achieved throughput, requests per second (RPS).

Figures 13-B and 13-C show memcached managed by **Octopus-Man** using the PID and Deadzone schemes, respectively. Octopus-Man dynamically decides the core resources allocated for memcached based on the measured QoS (95^{th} percentile query latency), migrating the job between brawny and wimpy cores, while the Static mapping (Figure 13-A) keeps memcached on brawny cores.

Octopus-Man's mapping algorithm using PID or Deadzone aims to adapt to the load and QoS changes. The PID control works by monitoring the QoS and minimizing the controller error; that is, keeping the measured QoS as close as possible to the QoS target. However, we notice that this particular

Mapping	QoS guarantee	QoS tardiness	Energy reduction (CPU)	Energy reduction (System)
Static (Brawny)	99.9%	1.06	–	–
Static (Wimpy)	**34.6%**	3.02	85%	40%
Octopus-Man (PID)	**61.3%**	1.66	49%	25%
Octopus-Man (Deadzone)	**99.8%**	**1.06**	**41%**	**15%**

TABLE II: Memcached QoS behavior and energy reduction

Mapping	QoS guarantee	QoS tardiness	Energy reduction (CPU)	Energy reduction (System)
Static (Brawny)	99%	1.29	–	–
Static (Wimpy)	**41%**	4.52	90%	51%
Octopus-Man (PID)	**45%**	3.56	74%	19%
Octopus-Man (Deadzone)	**91%**	**1.87**	**26%**	**9%**

TABLE III: Web-search QoS behavior and energy reduction

behavior causes many oscillations in the system, as shown in the middle plot of Figure 13-B. The large amount of oscillations in turn incurs severe QoS degradation, as shown in the top plot of Figure 13-B. On the other hand, Octopus-Man using the deadzone strategy can help mitigate this oscillation effect, as shown in Figure 13-C. We notice that from 0s to 250s, Octopus-Man adaptively adjusts the down threshold to minimize the oscillatory behavior. The final stable threshold found was $UP_{thr} = 0.8$ and $DOWN_{thr} = 0.1$. Compared to PID, Octopus-Man (Deadzone) with dynamic adjustment significantly reduced QoS violations.

Web-search — Figure 14-A shows the baseline **Static** all-brawny core mapping, Figure 14-B the PID control and 14-C the Deadzone scheme for the web-search workload. Similar to the memcached workload, the static mapping provides the best QoS, which is the upper bound for any dynamic mapping method, while Octopus-Man's mapping decisions performed by PID and Deadzone aim to monitor the QoS target and exploit the load fluctuations. The PID is more aggressive at performing task mapping to reduce energy consumption, but the excessive task switching activities have negative effects on the application QoS. Deadzone mitigates this issue by dynamically adjusting the QoS thresholds that trigger the re-mapping. This leads to less oscillation and better QoS.

Results Summary — Table II and Table III summarize the QoS behavior and energy reduction for memcached and web-search using different mapping options: Static all-brawny, Static all-wimpy, Octopus-Man (PID), Octopus-Man (Deadzone). We compare the QoS and energy consumption of each mapping scheme to the static all-brawny mapping as baseline. On each sampling interval we compute whether or not the measured QoS violates the QoS target. The *QoS guarantee* is the percent of the samples that the measured QoS is under the target (1 - QoS violations%). In case there is a violation, we compute the average *QoS tardiness* as the measured QoS value divided by the QoS target, quantifying how intense the QoS violation was.

As shown in Table II and Table III, Static all-wimpy cores cannot meet the required QoS for memcached and web-search. Octopus-Man using the PID control trades some power for better QoS, but as discussed earlier (Figures 13 and 14), the application QoS is degraded due to excessive dynamic task adaptations/mappings (2.5× to 10× more adaptations). Octopus-Man using deadzone is capable of delivering the best trade-off between QoS guarantee and energy reduction. Octopus-Man deadzone meets the QoS at 99.8% for memcached and at 91% for web-search with QoS tardiness close to the static all-brawny mapping. The CPU energy consumption is reduced by 41% for memcached and 26% for web-search. The measured dynamic system power is reduced by 15% for memcached and 9% for web-search.

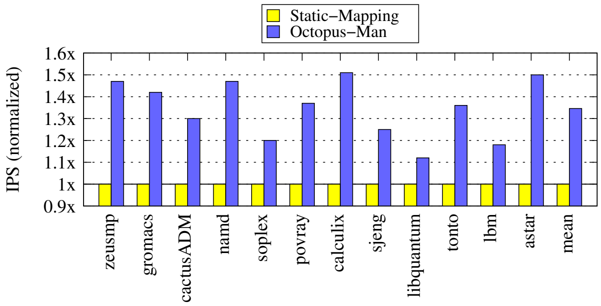


Fig. 15: Throughput improvement using Octopus-Man with batch job co-location

It is worth noting that, in contrast to Octopus-Man, prior work does not allow for QoS guarantees for latency-sensitive services since they are driven by either hardware performance counters (e.g., IPC, cache misses) [52] or CPU utilization [58] without providing explicit QoS management. For example, severe QoS degradation (95%-tile latency penalty of 254% and 5,989%) is reported for two datacenter workloads in prior work [58]. Such high latency degradations would likely violate QoS guarantees.

D. Improving Batch Throughput While Meeting QoS

Another use of the servers during low load periods in data centers is to co-locate batch jobs with service jobs on the same server. In this section, we evaluate the effectiveness of Octopus-Man (Deadzone) for improving the throughput of batch jobs while guaranteeing QoS of service jobs when batch jobs are co-located with web-search.

Throughput Improvement — Figure 15 presents the throughput improvement achieved by Octopus-Man compared to the Static mapping policy across 12 co-running batch applications from the SPEC CPU2006 benchmarks. Static keeps the web-search job on the brawny cores and the batch applications on the two wimpy cores. Octopus-Man, on the other hand, dynamically maps web-search and the batch applications across the wimpy and brawny cores. At a particular point in time,

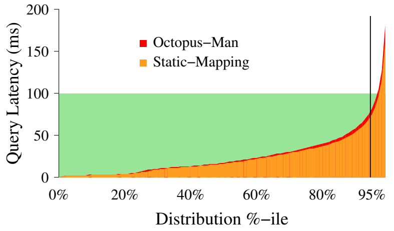


Fig. 16: Latency CDF of co-locating web-search and `calculix`

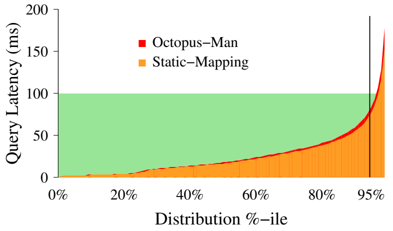


Fig. 17: Latency CDF of co-locating web-search and `lbm`

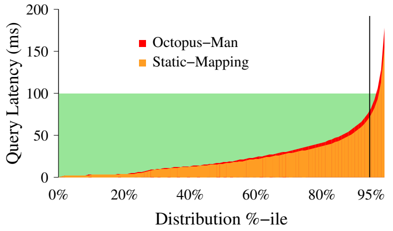


Fig. 18: Latency CDF of co-locating web-search and `namd`

the number of batch jobs running on the system corresponds to the number of cores not employed by the latency-sensitive application. We use aggregated IPS (instructions per second) to characterize the throughput of batch applications. Latency-sensitive applications are measured using application-level QoS metrics.

As shown in Figure 15, Octopus-Man always achieves higher throughput than the static mapping. On average, Octopus-Man achieves 34% throughput improvement for batch applications over the static mapping. The maximum (51%) throughput improvement is achieved when web-search is co-running with `calculix`, and minimal (12%) improvement when co-running with `libquantum`.

The throughput improvement is due to the fact that Octopus-Man can dynamically determine the minimum core resources needed by web-search for its satisfactory QoS, while using the rest of the resources for batch applications to maximize the throughput. For example, when web-search is experiencing very low load, Octopus-Man determines that 1 wimpy core is sufficient for its QoS target and reduces the core allocation for web-search accordingly, dynamically mapping batch applications to the other 3 cores to maximize throughput. When the load for web-search increases, Octopus-Man reallocates brawny core resources to web-search to ensure acceptable query latency during high/peak load.

QoS Guarantees — In addition to improving the batch throughput, we demonstrate that Octopus-Man delivers satisfactory QoS for web-search. Figures 16, 17, 18 present the cumulative distribution function (CDF) of web-search's query latency when it is co-running with 3 different representative batch jobs, respectively calculix, lbm, and namd. We selected benchmarks `calculix` and `lbm` to present because they represent compute-intensive and memory-intensive applications. Benchmark `namd` is selected because web-search experienced the worst QoS degradation when co-running with it. In each figure, the orange and red areas indicate the latency distribution achieved by the Static mapping and Octopus-Man, respectively. For example, in Figure 16, 60% of the queries are served within 23ms by Octopus-Man (red line) and 22ms by static mapping on Brawny (orange line). As long as the tail latency at the vertical line is within the green shaded zone, the QoS target is satisfied. As shown in these three figures, the query latency distribution achieved by Octopus-Man is very close to the static mapping, which always executes the web-search on 2 brawny cores. In all cases, the tail latency by Octopus-Man is shorter than the target, indicating satisfactory QoS.

VI. Related Work

The energy impact of warehouse-scale computers (WSCs) is large and has received much attention in recent years. As a result, there is a growing body of literature on the use of green energy [1], [20], [32], the overall energy footprint [48] and energy proportionality [4], [37], [41] in datacenters/WSCs. Our work focuses on improving the energy efficiency and proportionality of latency-sensitive applications in WSCs as well as the throughput of batch applications by mapping them to wimpy/brawny cores within a heterogeneous multicore architecture.

The energy and performance trade-offs between different types of general purpose processors are well-documented [13], [29]. Heterogeneity between servers [43] and specialization [34], [36] have been shown to produce efficient WSC designs. In WSCs, node-level and cluster-level techniques have been proposed [10], [22], [31], [42], [44], [45], [54], [55], [59], [61] to take advantage of architectural heterogeneity between servers and/or perform resource managment to improve efficiency in WSCs. In this work, we go a step further and exploit heterogeneity at the core-level within the server to deal with QoS and load-aware task scheduling.

Other work has explored combining cores of different capabilities within the server in WSCs. In [58], Wong et al. propose an architecture that combines commodity processors of varying capability as close together as a single board to cope with long-term changes in system load and improve energy proportionality. In contrast, our work utilizes architectures whose cores are tightly coupled and share memory. This allows for very fast task migration and responsiveness to changes while providing strict QoS guarantees.

Scheduling for heterogeneous multicore architectures has also been studied in prior work [8], [15], [28], [35], [52], [57]. Our scheduling approach is unique because it seeks to achieve multiple objectives, guaranteeing strict QoS/latency constraints for user-facing applications while improving throughput for batch applications in WSCs.

VII. Conclusion

In this work we describe a task management solution, Octopus-Man, that leverages a mixture of wimpy and brawny cores on core-heterogeneous systems to deliver improve energy efficiency and workload throughput in Warehouse Scale Computers (WSCs). Octopus-Man exploits the periods of low load common among latency-sensitive jobs, mapping those jobs to the least power-hungry processing resources that can satisfy their QoS requirements, thus greatly improving energy efficiency or freeing up high performance resources for other work.

We designed, implemented, and evaluated Octopus-Man on a real heterogeneous core platform (Intel QuickIA) experimenting on two different workloads – web-search and memcached – using realistic workload profiles. We show that Octopus-Man can improve energy efficiency over current scheduling policies by up to 41% for CPU power and up to 15% for full-system power, or alternatively that it can improve batch processing throughput by an average of 34%, all while adhering to QoS constraints for latency-sensitive jobs.

VIII. Acknowledgments

We thank our anonymous reviewers for their feedback and suggestions. This research was supported by Google and by the National Science Foundation under grants CCF-SHF-1302682 and CNS-CSR-1321047.

References

[1] B. Aksanli, J. Venkatesh, L. Zhang, and T. Rosing, "Utilizing green energy prediction to schedule mixed batch and service jobs in data centers," *ACM SIGOPS Operating Systems Review*, vol. 45, no. 3, pp. 53–57, 2012.

[2] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan, "FAWN: a fast array of wimpy nodes," *Commun. ACM*, vol. 54, no. 7, pp. 101–109, Jul. 2011.

[3] L. A. Barroso, J. Dean, and U. Holzle, "Web-search for a planet: The google cluster architecture," *Micro, IEEE*, vol. 23, no. 2, pp. 22–28, 2003.

[4] L. A. Barroso and U. Hölzle, "The case for energy-proportional computing," *Computer*, vol. 40, no. 12, pp. 33–37, Dec. 2007.

[5] L. A. Barroso and U. Hölzle, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," *Synthesis Lectures on Computer Architecture*, vol. 4, no. 1, pp. 1–108, 2009.

[6] L. A. Barroso and P. Ranganathan, "Guest editors' introduction: Datacenter-scale computing," *Micro, IEEE*, vol. 30, no. 4, pp. 6–7, 2010.

[7] N. Chitlur, G. Srinivasa, S. Hahn, P. K. Gupta, D. Reddy, D. Koufaty, P. Brett, A. Prabhakaran, L. Zhao, N. Ijih, S. Subhaschandra, S. Grover, X. Jiang, and R. Iyer, "QuickIA: Exploring heterogeneous architectures on real prototypes," in *HPCA '12*.

[8] J. Cong and B. Yuan, "Energy-efficient scheduling on heterogeneous multi-core architectures," in *ISLPED '12*.

[9] J. Dean and L. A. Barroso, "The tail at scale," *Commun. ACM*, vol. 56, no. 2, pp. 74–80, Feb. 2013.

[10] C. Delimitrou and C. Kozyrakis, "Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters," in *ASPLOS '13*.

[11] C. Delimitrou and C. Kozyrakis, "Quasar: Resource-Efficient and QoS-Aware Cluster Management," in *ASPLOS'14*.

[12] Electronic Educational Devices, "Watts Up PRO," 2010.

[13] H. Esmaeilzadeh, T. Cao, Y. Xi, S. M. Blackburn, and K. S. McKinley, "Looking back on the language and hardware revolutions: measured power, performance, and scaling," in *ASPLOS '11*.

[14] X. Fan, W.-D. Weber, and L. A. Barroso, "Power provisioning for a warehouse-sized computer," in *ISCA '07*.

[15] A. Fedorova, J. C. Saez, D. Shelepov, and M. Prieto, "Maximizing power efficiency with asymmetric multicore systems," *Commun. ACM*, vol. 52, December 2009.

[16] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ASPLOS '12*.

[17] E. Frachtenberg, "Holistic datacenter design in the open compute project," *Computer*, vol. 45, no. 7, pp. 83–85, 2012.

[18] G. F. Franklin, D. J. Powell, and A. Emami-Naeini, *Feedback Control of Dynamic Systems*, 4th ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[19] G. F. Franklin, M. L. Workman, and D. Powell, *Digital Control of Dynamic Systems*, 3rd ed. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1997.

[20] Í. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: managing datacenters powered by renewable energy," in *ASPLOS '13*.

[21] P. Greenhalgh, "Big.LITTLE processing with ARM CortexTM-A15 and Cortex-A7," White Paper, ARM, 2011.

[22] M. Guevara, B. Lubin, and B. C. Lee, "Navigating heterogeneous processors with market mechanisms," in *HPCA '13*.

[23] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury, *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[24] U. Holzle, "Brawny cores still beat wimpy cores, most of the time," *IEEE Micro*, 2010.

[25] T. Horvath, T. Abdelzaher, K. Skadron, and X. Liu, "Dynamic voltage scaling in multitier web servers with end-to-end delay control," *IEEE Trans. Comput.*, vol. 56, no. 4, pp. 444–458, Apr. 2007.

[26] V. Janapa Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, "Web search using mobile cores: quantifying and mitigating the price of efficiency," in *ISCA '10*.

[27] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-Way Multithreaded Sparc Processor," *IEEE Micro*, vol. 25, no. 2, pp. 21–29, Mar. 2005.

[28] D. Koufaty, D. Reddy, and S. Hahn, "Bias scheduling in heterogeneous multi-core architectures," in *EuroSys '10*.

[29] C. Kozyrakis, A. Kansal, S. Sankar, and K. Vaid, "Server engineering insights for large-scale online services," *IEEE Micro*, vol. 30, no. 4, pp. 8–19, Jul. 2010.

[30] R. Kumar, K. I. Farkas, N. P. Jouppi, P. Ranganathan, and D. M. Tullsen, "Single-isa heterogeneous multi-core architectures: The potential for processor power reduction," in *MICRO '03*.

[31] M. Laurenzano, Y. Zhang, L. Tang, and J. Mars, "Protean code: Achieving near-free online code transformations for warehouse scale computers," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-47. New York, NY, USA: ACM, 2014.

[32] K. Le, R. Bianchini, T. D. Nguyen, O. Bilgir, and M. Martonosi, "Capping the brown energy consumption of internet services at low cost," in *International Green Computing Conference*, 2010, pp. 3–14.

[33] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power management of datacenter workloads using per-core power gating," *IEEE Comput. Archit. Lett.*, vol. 8, no. 2, pp. 48–51, Jul. 2009.

[34] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi, "System-level integrated server architectures for scale-out datacenters," in *MICRO '11*.

[35] T. Li, P. Brett, R. C. Knauerhase, D. A. Koufaty, D. Reddy, and S. Hahn, "Operating system support for overlapping-isa heterogeneous multi-core architectures," in *HPCA '10*.

[36] K. Lim, D. Meisner, A. G. Saidi, and T. F. Wenisch, "Thin servers with smart pipes: Designing soc accelerators for memcached."

[37] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *ISCA '14*.

[38] P. Lotfi-Kamran, B. Grot, M. Ferdman, S. Volos, O. Kocberber, J. Picorel, A. Adileh, D. Jevdjic, S. Idgunji, E. Ozer, and B. Falsafi, "Scale-out processors," *SIGARCH Comput. Archit. News*, vol. 40, no. 3, pp. 500–511, Jun. 2012.

[39] C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, and S. H. Son, "Feedback control architecture and design methodology for service delay guarantees in web servers," *IEEE Trans. Parallel Distrib. Syst.*, vol. 17, no. 9, pp. 1014–1027, Sep. 2006.

[40] N. Madan, A. Buyuktosunoglu, P. Bose, and M. Annavaram, "A case for guarded power gating for multicore processors," in *HPCA '11*.

[41] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz, "Towards energy-proportional datacenter memory with mobile dram," in *ISCA '12*.

[42] J. Mars, L. Tang, and R. Hundt, "Whare-map: Heterogeneity in "homogeneous" warehouse-scale computers," in *ISCA '13*.

[43] J. Mars, L. Tang, and R. Hundt, "Heterogeneity in "homogeneous" warehouse-scale computers: A performance opportunity," *Computer Architecture Letters*, vol. 10, no. 2, pp. 29–32, 2011.

[44] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa, "Bubble-up: Increasing utilization in modern warehouse scale computers via sensible co-locations," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-44. New York, NY, USA: ACM, 2011, pp. 248–259. [Online]. Available: http://doi.acm.org/10.1145/2155620.2155650

[45] J. Mars, L. Tang, K. Skadron, M. L. Soffa, and R. Hundt, "Increasing utilization in modern warehouse-scale computers using bubble-up," *IEEE Micro*, vol. 32, no. 3, pp. 88–99, May 2012. [Online]. Available: http://dx.doi.org/10.1109/MM.2012.22

[46] D. Meisner, C. M. Sadler, L. A. Barroso, W.-D. Weber, and T. F. Wenisch, “Power management of online data-intensive services,” in *ISCA ’11*.

[47] A. K. Mishra, J. L. Hellerstein, W. Cirne, and C. R. Das, “Towards characterizing cloud backend workloads: insights from google compute clusters,” *SIGMETRICS ’10*.

[48] P. Ranganathan, “Recipe for efficiency: principles of power-aware computing,” *Communications of the ACM*, vol. 53, no. 4, pp. 60–67, 2010.

[49] V. J. Reddi, B. C. Lee, T. Chilimbi, and K. Vaid, “Mobile processors for energy-efficient web search,” *ACM Trans. Comput. Syst.*, vol. 29, no. 3, pp. 9:1–9:39, Aug. 2011.

[50] G. Ren, E. Tune, T. Moseley, Y. Shi, S. Rus, and R. Hundt, “Google-wide profiling: A continuous profiling infrastructure for datacenters,” *Micro, IEEE*, vol. 30, no. 4, pp. 65–79, 2010.

[51] S. Rivoire, P. Ranganathan, and C. Kozyrakis, “A comparison of high-level full-system power models,” in *HotPower ’08*.

[52] J. C. Saez, A. Fedorova, D. Koufaty, and M. Prieto, “Leveraging Core Specialization via OS Scheduling to Improve Performance on Asymmetric Multicore Systems,” *ACM Trans. Comput. Syst.*, vol. 30, no. 2, pp. 6:1–6:38, Apr. 2012.

[53] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, “Omega: flexible, scalable schedulers for large compute clusters,” in *EuroSys’13*.

[54] L. Tang, J. Mars, W. Wang, T. Dey, and M. L. Soffa, “Reqos: Reactive static/dynamic compilation for qos in warehouse scale computers,” in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, ser. ASPLOS ’13. New York, NY, USA: ACM, 2013, pp. 89–100. [Online]. Available: http://doi.acm.org/10.1145/2451116.2451126

[55] L. Tang, J. Mars, X. Zhang, R. Hagmann, R. Hundt, and E. Tune, “Optimizing google’s warehouse scale computers: The numa experience,” in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA ’13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 188–197. [Online]. Available: http://dx.doi.org/10.1109/HPCA.2013.6522318

[56] J. Treibig, G. Hager, and G. Wellein, “Likwid: A lightweight performance-oriented tool suite for x86 multicore environments,” in *ICPPW ’10*.

[57] K. Van Craeynest, A. Jaleel, L. Eeckhout, P. Narvaez, and J. Emer, “Scheduling heterogeneous multi-cores through Performance Impact Estimation (PIE),” in *ISCA ’12*.

[58] D. Wong and M. Annavaram, “Knightshift: Scaling the energy proportionality wall through server-level heterogeneity,” in *MICRO ’12*.

[59] H. Yang, A. Breslow, J. Mars, and L. Tang, “Bubble-flux: Precise online qos management for increased utilization in warehouse scale computers,” in *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, ser. ISCA ’13. New York, NY, USA: ACM, 2013, pp. 607–618. [Online]. Available: http://doi.acm.org/10.1145/2485922.2485974

[60] Y. Zhai, X. Zhang, S. Eranian, L. Tang, and J. Mars, “Happy: Hyperthread-aware power profiling dynamically,” in *USENIX ATC ’14*.

[61] Y. Zhang, M. Laurenzano, J. Mars, and L. Tang, “Smite: Precise qos prediction on real system smt processors to improve utilization in warehouse scale computers,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, ser. MICRO-47. New York, NY, USA: ACM, 2014.