ANT-Man: Towards Agile Power Management in the Microservice Era

Xiaofeng Hou*, Chao Li*‡, Jiacheng Liu*, Lu Zhang*, Yang Hu†, Minyi Guo*
* *Dept. of Computer Science and Engineering, Shanghai Jiao Tong University*
† *Dept. of Computer Science and Engineering, University of Texas, Dallas*
Emails: {xfhelen, chaol, liujiacheng, luzhang, myguo}@sjtu.edu.cn, Yang.Hu4@utdallas.edu
‡Corresponding author

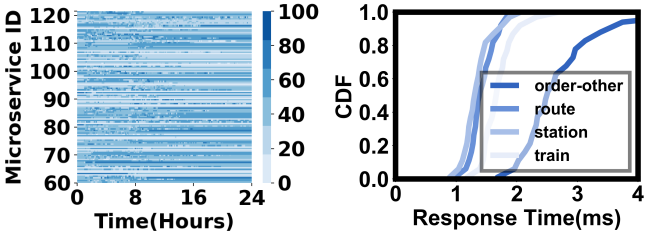
***Abstract*—The emerging trend of decomposing cloud applications into microservices has raised new questions about managing the performance/power trade-off of a datacenter at microsecond-scale. We introduce ANT-Man, an Auto, Native and Transparent power Management framework that can exploit fine-grained microservice variability for system efficiency. To achieve this, ANT-Man abstracts away two major sources of latency overhead in traditional hierarchical power management frameworks. First, ANT-Man proposes an auto power budgeting scheme for reducing the power coordination latency at the datacenter level. It can proactively determine the power budget tailored to each individual microservice. Second, ANT-Man proposes a native and transparent power control scheme to overcome the power configuration latency for each microservice. It enables super-fast power budget enforcement with nanosecond-scale performance scaling. Extensive experiments on our prototyped system show that ANT-Man could slash power consumption by 7.8∼43.5%, and in the meantime reduce the 95^{th} tail latency by 9.7∼12.5% compared to existing techniques.**

***Index Terms*—microservice, variability, power management**

I. Introduction

While online data-intensive (OLDI) applications once have largely monolithic software architecture, container technology has recently catalyzed the shift from the monolith to microservice [18], [28], [46], [62], [71]. Modern applications of this new architectural style are formed by a myriad of highly granular services that can be managed, tested, and scaled independently [46]. Some key attributes of microservice, such as domain-driven design and on-demand virtualization [3] make it fit nicely to today's scale-out cloud environment. Therefore, many Internet giants, including Google [71], Amazon [12], Microsoft [23], and Alibaba [4], are actively accelerating their adaption of microservice-based applications.

The significance of microservice goes beyond design flexibility. It also opens up a new door for improving efficiency in the cloud by exposing an *intra-application variability* (IAV) to datacenters. In this paper we use IAV to describe the complexity of microservices composing an application. From a scheduling point of view, IAV represents the differences in characteristics of short-lived tasks that occur on granular timescales. It puts an emphasis on two facts: 1) *spatial variability*, i.e., the disaggregated microservices exhibit different power-performance characteristics; 2) *temporal variability*, i.e., the behavior of a microservice changes dynamically at a fine-grained timescale. Figure 1 illustrates such variation across



(a) Power usage variability. (b) Execution time variability.

Figure 1: Intra-application variability (IAV). (a) Power demand of microservices running in a realistic datacenter [5]; (b) The cumulative distribution function (CDF) of microservices' response time in *TrainTicket* service [79]. Detailed evaluation methodology and results are shown in Sections III and V.

microservices that make up an application on production systems. Clearly, microservices not only present an uneven power usage profile but also exhibit a wide range of execution time at μs-scale. This observation prompts us to think about an important question: ***how can a well-managed datacenter actually make use of this unique variability given by the microservice architecture***? For example, a highly agile power management tailored to each service instance is expected to bring unprecedented performance-power trade-offs. In other words, IAV ultimately highlights an untapped energy-saving opportunities that we should not overlook.

Unfortunately, today's datacenter power management framework incurs high overhead, which makes it impossible to reap the full benefits offered by IAV. We mainly observe two broadly defined sources of overhead: *macro control latency* and *micro control latency*. The former one is usually related to a tedious global coordination process, i.e., the power budget monitoring and allocation in an iterative, hierarchical manner [54], [61], [72]. For example, in many mainstream datacenters, the delay can be up to 2∼10 seconds [20], [64]. The latter one is mainly related to a highly granular configuration process, i.e., the scaling of node performance at the chip level. Most of the power optimization schemes eventually rely on a software-enabled DVFS, which suffers from high latency of power state transition (6∼89 of microseconds) [14], [15].

In this work, we aim to overcome the above sources of latency overhead and build a truly agile power management

framework. In this way, one can operate datacenters in a highly efficient, IAV-aware manner. Although sub-microsecond scale power management seems not necessary for monolithic applications, lacking such mechanism may cause imprecise power allocation and energy waste for short-lived tasks (which often finish execution within a few hundreds of microseconds [25]). Recently, the "killer microseconds issue" [17] has drawn people's attention to highly granular system management. However, little work has been done in terms of eliminating the above latency overheads for short-lived (10$\sim$100μs) tasks.

We propose ANT-Man, an **A**uto, **N**ative and **T**ransparent datacenter **Man**agement framework for fully exploiting the efficiency optimization opportunity exposed by IAV. First, Our framework features a novel auto power budgeting (APB) technique for capturing microservice characteristics with mitigated macro control overhead. It allows one to proactively orchestrate power allocation based on workload criticality, thereby avoiding the time-consuming global control loops. In addition, our framework adopts a novel native performance scaling (NPS) technique for eliminating the micro control overhead. The NPS module can achieve super-fast performance scaling by enabling the system to directly invoke on-chip voltage regulators. Finally, ANT-Man employs a transparent mapping mechanism to gracefully coordinate the above efforts in a highly efficient, user-transparent manner.

Summarily, we make the following contributions:

- We demonstrate power saving opportunities introduced by intra-application variability (IAV) through detailed microservice characterization. We show that traditional hierarchical datacenter power management frameworks cannot exploit IAV due to high latency overhead.
- We propose ANT-Man, a novel power management framework that can fully exploit IAV under the complexity of microservices in the cloud. ANT-MAN provides necessary system abstractions and automation processes to enable highly efficient sub-microsecond performance scaling.
- We build a prototype of ANT-Man. We verify its effectiveness with the best benchmark suites available at this moment, including an industrial microservice suites (*TrainTicket*) as well as an academic microservice benchmark (*DeathStarBench*). We show that ANT-Man can save power by 7.8-43.5% compared to the state-of-the-art power optimization of OLDI applications.

The remainder of the paper is organized as follows. Section 2 introduces more backgrounds. Section 3 details the opportunities and challenges. Section 4 proposes ANT-Man. Section 5 presents experimental methodologies and results. Section 6 discusses related work. Finally, Section 7 concludes the paper.

II. Backgrounds

A. Microservice Architecture

Microservice architecture is taking cloud computing by storm as the preferred style for developing highly scalable and modular applications [3], [12], [46], [62], [71]. It presents a code-heavy monolithic application as a suite of self-contained process-level microservices, as shown in Figure 2. Built based on the concept of share-as-little-as-possible, microservice leverages bounded context, which is a domain-driven design to restrict service range. The bounded context refers to the coupling of a microservice and its associated data as a single closed unit with minimal dependencies [62]. Microservices in different bounded contexts communicate with each other through light-weight mechanisms such as remote procedure calls (RPC) [7]. Unlike traditional service-oriented applications (SOA) orchestrated by a message-passing middleware, microservice implements an API layer as a service-access facade (Figure 2). Generally, SOA presents itself as a grid with complex links of services, while microservice architecture is more like a bipartite graph [39], [62], [68], [69]. In the graph, the API layer acts as a service-access portal and the service layer contains massive loosely-coupled microservices.

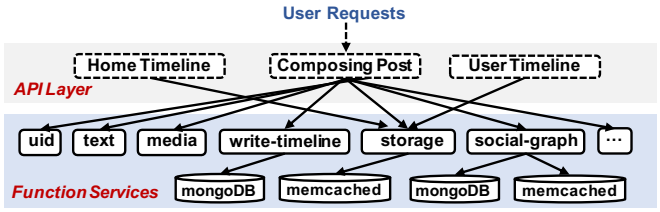


Figure 2: Dependency graph of *Composing Post* service in *SocialNetwork* application [31].

B. Cloud Power Management

Cloud datacenters need appropriate power management strategies to reduce energy consumption while ensuring Service-Level Objectives (SLO) [22], [62], [65]. The SLO determines specific service level targets (e.g., response time of the application). In general, an SLO-aware power management scheme can save energy by controlling the application's execution to just meet the deadline (SLO target). In Section 7 we provide a detailed comparison of different strategies.

In this paper, we envision that datacenter power management frameworks need to be *cloud-native* [1], [32], which means that the system should take into account the nature of the cloud application. Today, the "cloud native" initiative [1] is gaining popularity and microservice architecture has become the core of it. However, many architectural features and system mechanisms for energy efficiency are not built specifically to run in the emerging microservice environment. When facing a myriad of short-lived microservices, ***traditional power management needs to be re-examined at different layers to meet the new latency requirements***.

III. Opportunities and Challenges

In this section, we conduct in-depth analysis using the state-of-the-art microservice benchmark suites from both academia and industry. Our characterizations disclose the intra-application variability (IAV) issue brought by the microservice and the ensued energy-saving opportunities. We also demonstrate the challenges of managing power for microservices which often serve a request within a few microseconds.

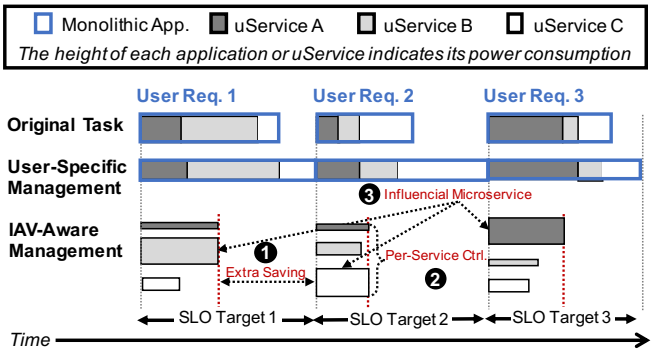


Figure 3: Energy saving opportunities brought by IAV in the microservice environment.

A. Opportunities: IAV-Aware Power Management

1) Overview of the Opportunity: Using a schematic example in Figure 3, we show how IAV brings additional latency-saving and energy-saving opportunities. In this example, a monolithic application is triggered by three separate user requests, resulting in different execution time and energy consumption.

We assume that the monolithic application could be decomposed into three *independent* pieces of microservices, namely μService A, B, and C. The system uses the execution time of the longest microservice as the new deadline for each service. These microservices have little dependencies and would not cause interference with each other. Since microservices execute intrinsically faster, they present longer latency slacks between the new execution deadline and the target SLO (extra energy saving opportunities)❶. One can reduce power through per-service management mechanisms that fine-tune the core frequency of each microservice ❷. In addition, some microservices play a more important role. We define *influential microservices* ❸ as those that are most likely to result in SLO violation of the associated request under power capping. It is desirable to track and locate microservices that affect the dynamic power consumption most when user request changes.

2) Analyzing the Inter-Application Variability: To understand how IAV affects power management decisions, we experiment with *SocialNetwork* from the open-source microservice benchmark *DeathStarBench* [31]. This is a unidirectional social network implemented with loosely-coupled microservices. It serves three kinds of requests including reading home timeline (*Home*), reading user timeline (*User*) and composing posts (*Post*). We deploy *SocialNetwork* with *docker swarm* [13] on a local server cluster. We present detailed hardware specifications in Section 5. We write *Python* programs to automatically access user requests via a manager node. By analyzing the request-tracing data with *Jaeger* [9], we can obtain the response delay of each microservice. We repeat our experiment for 1000 times and report the average results in Figure 4.

Observation 1: *Microservices are differently sensitive to power budget variations*. Figure 4-(a) shows the average response time of the evaluated microservices. It is evident that the performance of some microservice such as *uid* can be more sensitive to power capping and frequency scaling.

Observation 2: *Microservices' performance sensitivity to power budget relates to request types.* We evaluate how different request types (i.e. *Home, User, Post*) may affect the responsiveness of microservice *storage*. In Figure 4-(b), it is clear that the delay of *storage* is more sensitive to frequency change when serving the *User* request type.

Observation 3: *Microservice's performance/power correlation is tied to load levels*. In Figure 4-(c), the slope indicates the performance-power correlation under different number of concurrent requests (Low: ∼10, Medium: ∼200, High: ∼500). As we can see, a higher degree of request concurrency usually leads to more performance sensitivity to power capping.

Importantly, the above observations are made in a highly granular task execution environment. In addition, each microservice may have different impact on the SLO of the entire application. In the following we show the challenges of designing an IAV-aware power management framework.

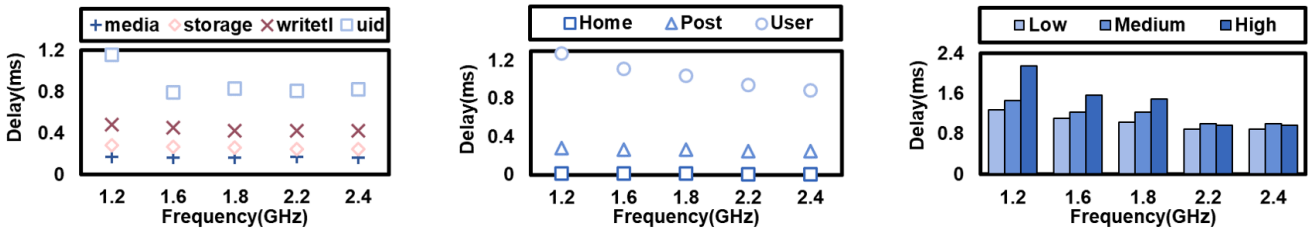
B. Challenges: Cross-Layer Control Latency

1) Overview of the Challenge: From the viewpoint of power management, microservice presents a treasure (IAV) not easy to attain. The reduction in service execution time has outpaced the improvement of power control speed. Although a processor can use on-chip voltage regulators (VRs) to change the voltage to a new level with tens of nanoseconds [44], the system-level power management can be several orders of magnitude slower. Figure 5 shows the major sources of power management latency in today's power-constrained datacenter.

Macro Control Latency (Tens of seconds): At the datacenter level, global power management aims to coordinate power demand across a large number of servers or applications. It spends a long time on collecting power consumption data, monitoring server utilization, and determining the optimal power allocated to each node [36], [61], [72]. It involves many control loops at multiple levels of the datacenter power management hierarchy. This iterative decision-making process may waste over 10 seconds [20], [72].

Micro Control Latency (10-31 milliseconds): Most of the available interfaces or policies of performance scaling are purely software-enabled [14], [15]. Once local servers receive the power operation commands, it takes over 10 milliseconds for the operating system to resume the runtime conditions of active cores [14], [15]. After that, it could waste tens of microseconds on conveying the commands by switching from user mode to kernel mode and calling the associated drivers (for voltage/frequency scaling) [19], [47], [69].

2) The Need for μs-Scale Control : To substantiate that existing power management schemes are not prepared for microservice, we compare the service time of a representative *Memcached* microservice [11] and the control latency of power management tools integrated in servers. We measure the DVFS transition time as well as response time of the *Memcached* microservice on a 40-core server (Intel Xeon Silver 4114) running Ubuntu 16.04. By setting the power management mode as *OS Demand Based Power Management* and turning off



(a) Service type. (b) Request type. (c) Request number.

Figure 4: The Intra-Application Variability of *SocialNetwork* Application in *DeathStarBench* Benchmark.

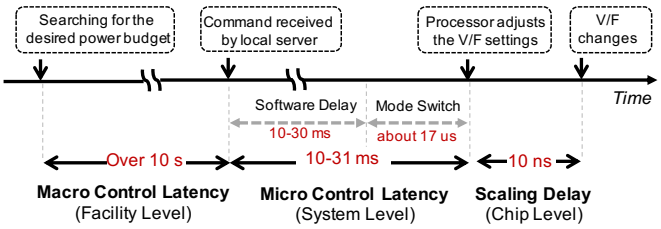
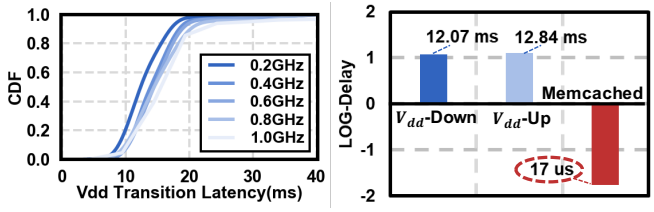


Figure 5: Timeline showing the main latency sources in conventional power management.



(a) Transition delay of decreasing V_{dd} at various levels. (b) V_{dd} transition delay vs. *memchached* execution time.

Figure 6: Analysis of DVFS transition delay.

turboboost technology, we can change the frequency of each cores via *cpuinfo_max_freq*. We deploy *Memcached* service on the server with container. We write a C program to access the *Memcached* service and change the server's frequency. The program also record the frequency transition delay and count the completed request number of *Memcached*.

In Figure 6-(a), we decrease the frequency from 2.2GHz in a step of 0.2GHz. Figure 6-(a) shows that the transition time is between 10-30 milliseconds. Figure 6-(b) further shows that the difference between the average transition (scaling up and scaling down) time of DVFS and *Memcached*'s execution time can be several orders of magnitude. (13 milliseconds for DVFS v.s. 17 microseconds for Memcached).

C. Summary of Design Consideration

Today's datacenter power management approaches are being left behind while the upper-layer applications start to reap the versatility benefits of microservices. A well-managed cloud infrastructure needs to abstract away enough of the underlying power control complexities to easily adapt to the microservice environment. Specifically, two questions must be addressed: 1) How to effectively allocate power and fully utilize the energy-saving opportunities brought by IAV; 2) How to get rid of the power control overhead and tackle the challenge of μs-scale management to fit to users' fleeting requests.

IV. The ANT-Man Framework

To address the above issues, we propose ANT-Man, an **A**uto, **N**ative and **T**ransparent **Man**agement framework that is built to be cloud-native. Figure 7 gives an overview of ANT-Man. It highlights three function modueles detailed below:

1) **A**uto-Power Budgeting (APB): Our design starts by exploiting the optimization opportunities at the application layer. The APB module aims to quickly capture the IAV by analyzing the performance-power features and the target SLO. Meanwhile, it proactively determines differentiated power budget for each microservice rather than iteratively compares the power usage with the total budget to mitigate the macro control latency.
2) **N**ative Performance Scaling (NPS): The main purpose of the NPS scheme is to eliminate the power management latency at the bottom execution environment (micro control latency). It implements a daemon process to directly regulate on-chip VRs through bypassing the OS-associated management. It defines key model-specific registers to facilitate fast on-chip voltage transition.
3) **T**ransparent Mapping: ANT-Man coordinates APB and NPS through a user-transparent interaction mechanism. It implements a software tag (s-tag) for tracking the varying characteristics of microservice. Meanwhile, it deploys a hardware tag (h-tag) for per-microservice power control. Eventually, it allows a datacenter to oversee microservices and fast manage their power for better design tradeoff.

Note that ANT-Man's agile power management approach is not limited to only the microservice programming style. It can be applied to multi-threaded or massage-passing applications. Adjustments will need to be made, such as re-defining appropriate software tags in that particular computing environment.

A. Auto-Power Budgeting

The APB scheme splits traditional power budgeting into two sub-tasks: 1) Fast Model Generation, which quickly builds the model for expressing intra-application variability; 2) Differentiated Power Allocation, which automates the power budget allocation for influential microservices.

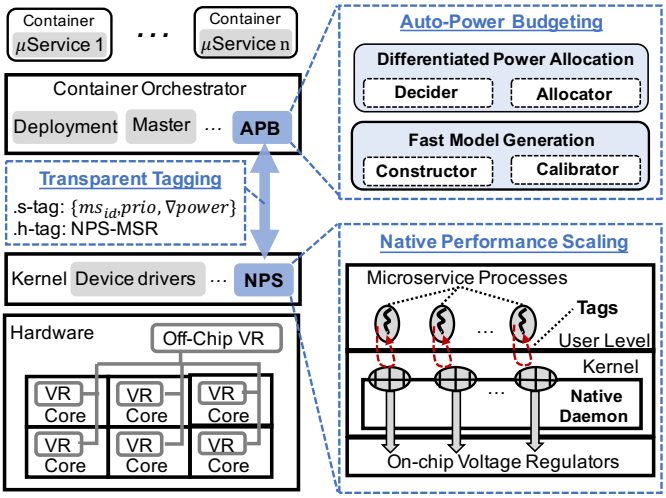
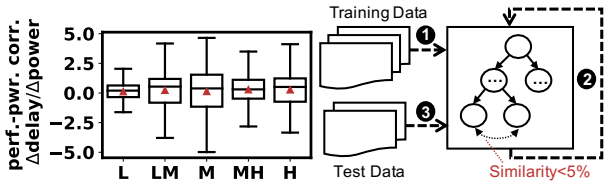


Figure 7: An overview of ANT-Man's key components.



(a) Dist. of perf./pwr. patterns under five load levels.

(b) Decision tree based model construction.

Figure 8: Automated model generation.

Fast Model Generation: Establishing a set of performance-power models in a highly dynamic environment with massive microservices can be challenging. It is impractical to specify a unique model for each service and maintain an exhaustive list of models covering all their states. Nevertheless, we observe that the load level of a microservice can be an effective indicator for its performance-power correlation. According to the Central Limit Theorem [6], we find that the performance-power correlation of a microservice at different ranges of users' load can achieve a confidence level of 95%. As shown in Figure 8-(a), we measure the performance-power correlations under different load levels using web service statistics [63] and present the distribution of the slope of the performance-power curves. We can note that the performance-power correlation patterns under given input load level are highly similar, where the input loads only have five levels. Therefore, we assume that the performance-power correlation of a microservice can be expressed as a piecewise function of users' load state. By constructing the model with finer-grained load levels, one can approximate the performance-power correlation with acceptable or very high accuracy in a fast and efficient manner.

We leverage a widely used learning-based technique, namely decision tree algorithm [45] to automate the model construction process as shown in Figure 8-(b). We extract training data and test data from real-world running logs. The log file consists of time series of data pairs

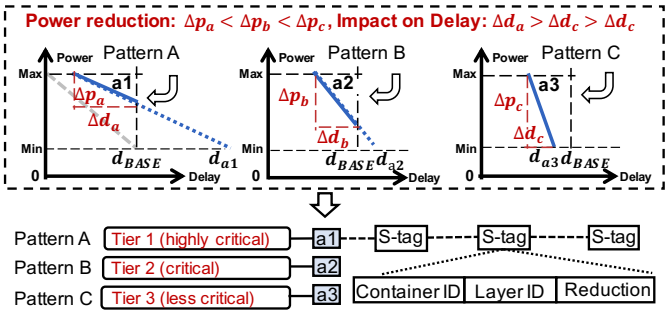


Figure 9: The mechanism for ranking microservices.

$\langle response\_time, frequency, user\_load \rangle$. It shows the response time of requests under different voltage/frequency settings and various user loads. The training data is first fed into a module❶ running the algorithm; then the algorithm interactively divides❷ the training data into two branches until the target curves on the two leaf nodes become close, namely, the difference (expressed as the mean error) between them is less than 5%. After that, the test data is used to verify❸ the model. We choose a decision tree algorithm for several reasons. First, it does not require prior knowledge about the dependencies between target profiles and load quantity, making it applicable for services where load changes frequently. Second, the decision tree has been shown to be especially effective in classifications. Third, it supports fast model calibration.

Our model constructor is designed to self-optimize itself over the lifetime. To be specific, a calibrator continuously compares the estimated values with the actual execution data. If the error exceeds a given margin (e.g., 5% [6]), it will rebuild the leaf node. Once a service changes greatly or a completely new service comes, it will progressively update its model in the following iterations. The above process only introduces minor overhead. For updating a leaf node, the program complexity is $O(1)$ since it maintains an up-to-date log for reconstruction. The algorithm complexity for rebuild a model (adding a new tree) is $O(n)$ [26] as it involves data acquisition and processing. However, the latter process rarely happen in a real cloud.

Differentiated Power Allocation: ANT-Man adopts a differentiated power allocation scheme, as depicted in Figure 9. The process of power budget allocation is mainly driven by the power/performance characteristics as well as the target SLO (e.g., *d_BASE* in the figure). The slope of a line indicates the performance-power correlation, and the endpoint on the x-axis is the worst-case execution time of the microservice running at the lowest speed. In other words, a diagonal dashed line actually represents the case that the response delay of microservice changes from 0 to the maximum allowable value (the power demand declines from the most to the least). It is a critical boundary that one can achieve the most aggressive power reduction without violating the SLO. In both *Pattern A* and *Pattern B*, it ultimately worsens the SLO after aggressive power reduction. In contrast, it is safe to reduce the power of services with *Pattern C* since they never exceed the worst delay. Compared with *Pattern B*, the slope in *Pattern A* is less than

the critical baseline, which means microservices with *Pattern A* are more sensitive to power reduction. Overall, microservices with *Pattern A* is more critical than *Patterns B* and *C*.

We determine the power assignment among massive microservices based on their patterns. As shown in Figure 9, when a power violation occurs, ANT-Man will first de-allocate services from a lower-influence (less critical) queue. Meanwhile, we analyze the influence within the same queue by comparing their power reduction potential factor which is calculated as the slope multiplying their absolute power usage. To sum up, if there is a power peak, ANT-Man will reclaim the power from the lowest-influence services.

It is worth mentioning that ANT-Man can work together with microservice scheduling schemes. For example, by smartly co-locating microservices with similar priority to certain CPU core, one can improve the effectiveness of differentiated power allocation. We leave this question for our future research.

B. Native Performance Scaling

The goal of the native performance scaling (NPS) module is to quickly change the performance state of the actual running process after budgeting the total power for each microservice. It contains an NPS daemon process and a system call in the kernel to activate the on-chip Voltage Regulators (VRs). It also defines an NPS_MSR (NPS model-specific register) to facilitate the call of the NPS daemon process.

NPS Service Console: As shown in Figure 11, systems have implemented several power management governors such as *userspace* [75] in the kernel layer. The mechanism behind these governors leverages the *acpi-CPUFreq* interface [75] to control the on-chip VRs via specific power management MSRs (PM_MSR) such as Intel IA32_PERF_STATUS_MSR [2]. Then the on-chip VRs change the frequency of the processor. With reduced frequency, the voltage can be decreased, leading to a cubic drop in power. Existing techniques often incur over 10 milliseconds long period of resuming runtime information [14], [15]. Instead, NPS directly controls the on-chip VRs without the above complex procedures. It implements the NPS_MSR to facilitate V/F (voltage/frequency) scaling.

Customized MSR: We devise NPS_MSR. It is defined per hardware thread, aiming to associate each microservice with high-speed on-chip VR transition. As shown in Figure 10, it is a hardware register with 4 bits. The last bit is a dirty bit that reflects whether we need to update current V/F setting. A dirty bit of 1 means current V/F state needs to update. The NPS_MSR register uses the remaining bits to convey the desired V/F setting. It leverages only 3 bits to inform the target V/F setting since the processor only supports a few discrete V/F configurations. As the OS or VM orchestrator swaps the thread of a microservice onto a core, it will check the microservice's requirement on V/F setting. It writes the dirty bit with 1 if this microservice requires different V/F settings, and the target V/F setting is loaded into bit[4:1]. Figure 10 shows the implementation of NPS_MRS on Intel microprocessor. We can use the reserved bits of IA32PQR_ASSOC_MSR in its Cache Allocation Technology [8] to define the NPS_MSR.

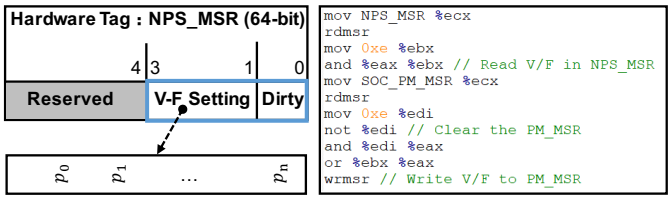


Figure 10: The hardware structure of NPS_MSR.

Daemon Process: The NPS module runs a daemon process, namely a background process on the local server side. This process starts at the boot time of the system and stops when the system terminates. It keeps inquiring the dirty bit of NPS_MSRs. If the dirty bit is 0, it keeps listening. Otherwise, it will call the *sys_nps_pm* to execute the V/F transition operation. As shown in Figure 10, the *sys_nps_pm* writes the target V/F setting (bit[4:1] of NPS_MSR) into the PM_MSRs. After that, the underlying hardware transfers the clock and changes voltage to the target. Comparing to existing implementations such as Xen power manager [14], [15], NPS can bypass the micro execution latency through fast V/F transition.

C. Transparent Mapping Mechanism

The transparent mapping mechanism is the key to build the cross-layer power management abstraction of ANT-Man. It gracefully coordinates the function of APB and NPS modules by placing and capturing special identifying tags.

Tagging Mechanism: Every on-the-fly microservice in our design is associated with a light-weight software tag (s-tag) and a hardware tag (h-tag). The h-tag is implemented with the NPS_MSR depicted in Section 3.4. Its main purpose is to speed up the per-microservice V/F transition. ANT-Man leverages s-tag to track the execution status as well as power assignment of massive microservices. Compared with h-tag, implementing s-tag is a little bit more tricky. A microservice is running as a container and then these containers run on a cluster of hosts. Due to the huge quantity of containers, cluster operators always leverage an orchestration system to configure their execution and obtain the information of these containers. Thus, one can implement s-tag based on the existing container orchestration system used like *docker swarm* [13] and *Kubernetes* [10]. It only requires minor software modification. Taking Google's kubernetes as an example, we can assign a unique s-tag to a container by describing a specific deployment in the *YAML file* and monitor its load variance through adding extra observing properties with *kubectl get pods*.

In detail, in order to distinguish different microservices, a unique s-tag is assigned to each microservice by the centralized container manager like *kubernetes* [10] when it starts execution. In other words, an s-tag is always bounded to a specific microservice. It has been written into the docker's data region when initiating the associated docker. The s-tag travels along with the microservice during its whole lifetime. Once a microservice completes its execution and exits, part of the s-tags are freed to minimize space. Some s-tags need to be preserved to take advantage of the temporal locality to accelerate control.

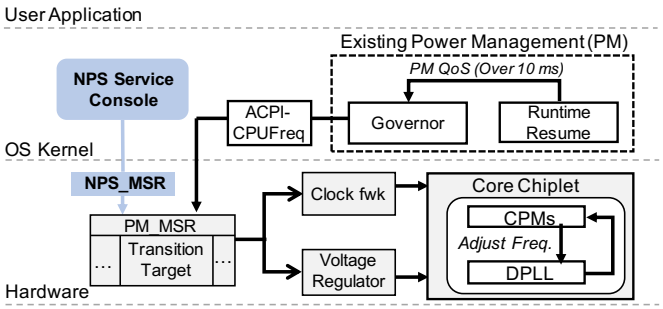


Figure 11: The main components of NPS.

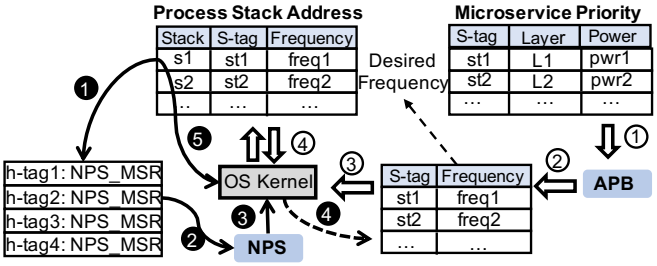


Figure 12: The data flow and invocation path of NPS.

For example, once a influential microservice restarts after last termination, it will be assigned with the same s-tag to avoid repeating the same performance-power modeling process.

Control Flow: Figure 12 illustrates the control flow in the ANT-Man framework. Before each microservice starts running, ANT-Man has computed a power allocation table through the input of a influence table①. Each row in the table is associated with a microservice by attaching with the s-tag. ANT-Man transfers the power table into a frequency table② with the model offered by the APB module. The frequency table is sent to the operating system③ to make scheduling decisions. OS writes the frequency into the stack space of each microservice in accordance with their s-tag④ when scheduling it onto the CPU. Thus, OS associates each microservice with their logic thread through s-tag. Before a microservice starts execution, OS kernel assigns❶ a hardware thread to it and triggers❷ the V/F transition of this hardware thread through NPS_MSR. As NPS keeps inquiring the NPS_MSRs, it captures the changes and re-configures❸ the V/F of the core running the hardware thread to desired setting. Once changing the V/F setting, the NPS exits❹ intermediately. Eventually, the stack space of target microservice is called❺ onto OS. Thus, the microservice eventually executes under the desired frequency.

V. Experimental Methodologies

We build a proof-of-concept system and implement a prototype of ANT-Man on real machines. Specifically, we realize our NPS scheme on a Dell PowerEdge R730 server. The server has a 6-core CPU (Intel Xeon E5-2620, 2.4GHz) with Ubuntu 14.04 installed as the operating system. The processor of the server supports per-core DVFS with operating frequencies from 1.2GHz to 2.4 GHz at the interval of 0.1GHz.

Table I: Testbed configuration in our experiment.

Node Role	Running MS	Description
Swarm manager	Zipkin/Jaeger	Providing web interface for observing.
Power worker	Observed MS	Observing MS at various V/F settings.
Normal workers	Other MS	Excluding other influence factors.

Cluster's Server Configurations	
Cluster	4 worker nodes (total 24 cores) + 1 manager node.
Server	Dell R730 6-core, 2.4 GHz Intel Xeon CPU E5-2620 v3.
Host OS	Ubuntu 14.04.31-generic kernel, docker 18.06.1-ce.

Table II: The evaluated power management schemes.

Scheme	Brief Description
ANT-Man	Our proposal.
PC	Application-specific mgmt. considering power
TC	Application-specific mgmt. considering performance (SLO)
CFP	Application-specific mgmt. considering power-perf behavior
CAP	Application-unaware power management.

With Linux tool *turbostat*, we can read the dynamic power of every server. We pin the container running the observed microservice to a specific core. Meanwhile, we leverage the reserved bits in the IA32_PQR_ASSOC_MSR [8] as our NPS_MSR. With library function *daemon()*, we implement the NPS daemon process in the kernel. We also add the *sys_nps_pm* into the reserved system call table. For comparison, we also use the *userspace* governor, which presents the conventional power management deployed in Linux.

We test an industrial microservice application named *TrainTicket* [79] and an academic microservice benchmark named *DeathStarBench* [31]. To establish the performance-power model, we respectively run them on a cluster as shown in Table I with *docker swarm*. The cluster contains 1 manager node and 4 worker nodes (in Table I) with 100 watts nameplate power per server. All the servers are connected to a FAST FSG116 network switcher to ensure high-speed network among microservices. The manager node provides web interfaces for gathering timing data of each microservice. We write several *Python* programs to respectively send requests accessing the *Advanced Search* service in *TrainTicket* and *compose-post* service in *SocialNetwork*. We leverage the time-tracing tools like *Zipkin* and *Jaeger* to collect each microservice's response delay. For power and frequency collection, we use *turbostat*.

We compare our design with other four kinds of the present power management schemes as summarized in Table II. Among those, Capping (CAP) is a representative peak power management technique similar to prior work [57], which only scales down the overall servers' active power to shave peak power. Power-driven Capping (PC) represents a group of application-aware schemes that only allocates power in accordance with individual power consumption [67]. Instead, Time-driven Capping (TC) is an approach based on the response delay or tail latency of each application [74]. Both PC and

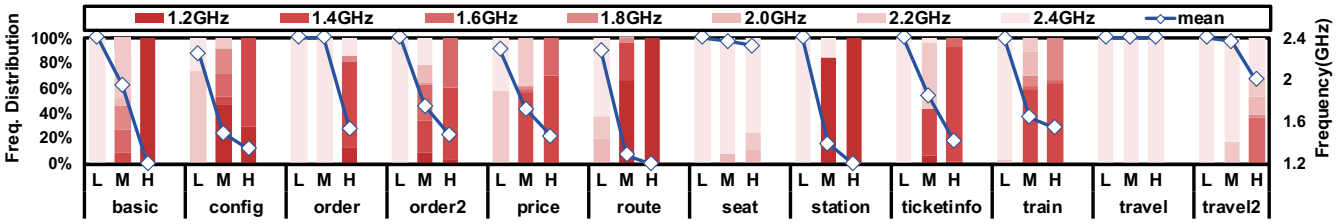


Figure 13: The frequency distribution of different microservices under L(Low), M(Medium) and H(High) traffics.

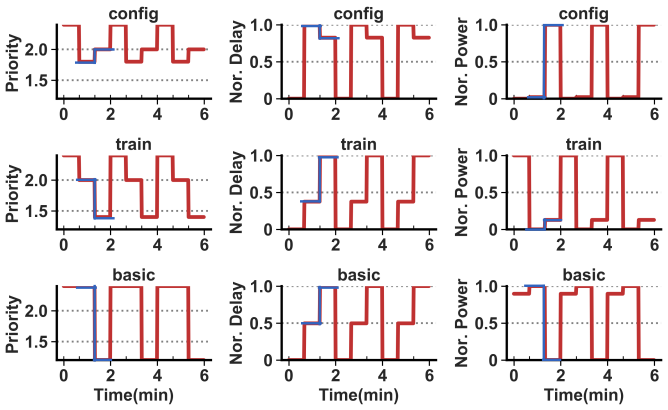


Figure 14: The influence of load levels.

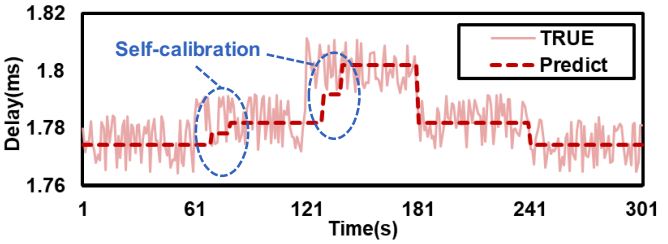


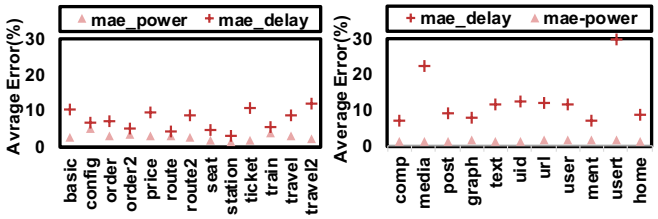
Figure 15: The self-calibration process of ANT-Man.

TC overlook the association between power variance and performance. Few existing methods like CFP also consider the profile of performance and power, but they neglect the dynamic changes of application execution states. All these mechanisms are too coarse-grained to manage power at microsecond scale.

VI. Evaluation Results

A. Evaluating the Auto-Power Budgeting

Effectiveness Analysis: It is highly desirable if ANT-Man could assign flexible V/F for different microservices at different load levels. In Figure 13, we present the frequency distribution of 12 microservices associated with *Advanced Search* at low (L), medium (M) and high (H) load levels. In the figure, the stacked bar plot presents the distribution of different execution frequency and the line plot depicts the average execution frequency of different microservices. The result shows that *travel* microservice always executes with higher frequency due to its higher priority compared to the others. In contrast, *station* frequently runs at the lowest frequency. Most of the time, the average executing frequency changes as load level varies.



(a) *TrainTicket.* (b) *SocialNetwork.*

Figure 16: The results of accuracy analysis.

Note that load levels can have a great impact on microservice power demand. To illustrate this, we show detailed running statistics of *config, train* and *basic* in Figure 14. We choose these three microservices since their priority value reversely changes with load variation. We can see that the way power demand changes in *train* is different from *config* and *basic*. As the priority value decreases (a result from load increase), *train* incurs more execution delay due to performance scaling. However, its power demand increases instead as shown in the figure. It is mainly because the impact of load increase outperforms the impact of performance scaling for *train*. In our design, ANT-Man can automatically identify the optimal power allocation strategy based on the priority of each microservice.

Accuracy Analysis: It is critical to examine whether our model constructor can calibrate itself with time-varying load levels. In Figure 15, we demonstrate a piece of self-calibration process in our framework. Taking *config* as an example, we compare the response delay of obtained from the performance-power model and a real running log. We feed ANT-Man with an initial performance-power model when it starts. Therefore, at the beginning of the experiment, the response delay computed with performance-model is always equal to the actual monitored value. When the first load surge comes at the 61^{st} second, our system detects the difference between the actual delay and the computed delay. Consequently, it takes a few seconds to rebuild the performance-power model (self-calibration). The reconstruction also happens at the 121^{st} second. Remarkably, in the following timestamps when the average traffic drops at the 181^{st} second and the 241^{st} second, no calibration is performed. This is because our system has already built a decision tree with necessary leaf nodes associated with different load levels.

ANT-Man maintains a fairly accurate performance-power

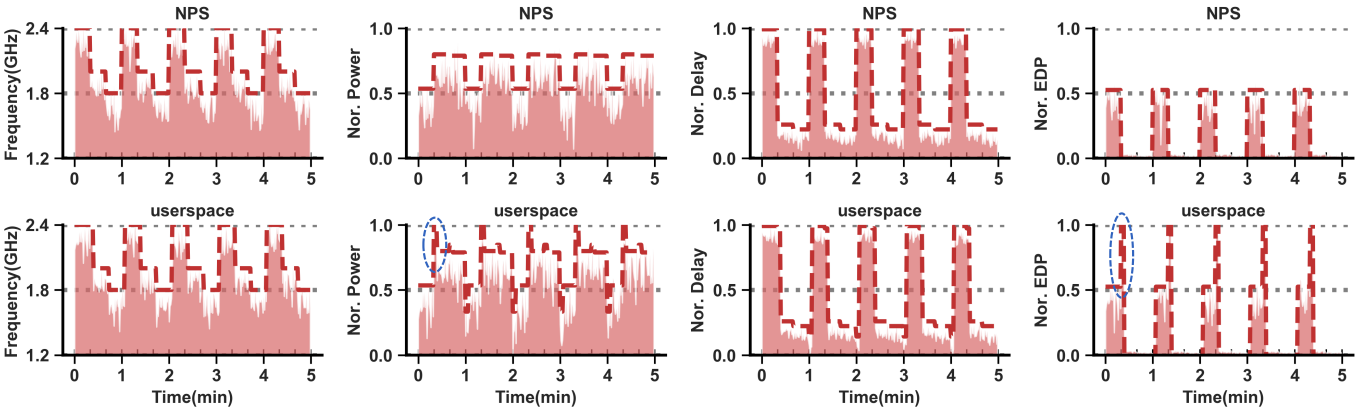
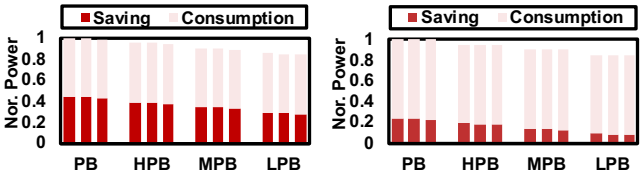


Figure 17: The comparison between NPS and *userspace* governor.



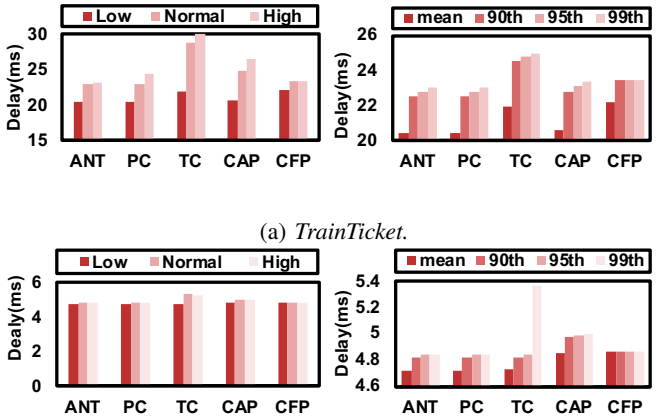
(a) *TrainTicket.* (b) *SocialNetwork.*

Figure 18: The results of energy saving.

model to instruct microservice-level power assignment. Figure 16 shows our investigation of model generation accuracy. We compute the mean absolute error (MAE) [21] between the predicted value and the measured value. We normalize our results to the measured value. We observe that the MAE with respect to power demand is always less than 2%. The MAE regarding to response delay is roughly around 10%. We do see inaccurate predictions (e.g., the result of *media* and *usert* exceeds 20%). The reason for this inaccuracy is that it is difficult for us to collect the short response delay. One may leverage other micro-architectural metrics such as Million instruction per second (MIPS) to improve accuracy.

B. Testing the Native Performance Scaling

Fast power regulation of each microservice is important. This part examines the impact of DVFS speed on ANT-Man at the server level. We compare our NPS module with the *userspace* governor [75] in the existing kernel. We trigger frequent V/F adjustment through changing the load every 30 seconds. Figure 17 shows our results. The solid lines present the actual variation and the dotted lines plot the overall trend. It is evident that NPS can always transfer voltage and frequency fast. However, by comparing NPS with *userspace* governor, we can see that there is a obvious transition delay of *userspace* governor (as highlighted by the circle in the second and forth columns). The transition latency is about 20 milliseconds long. As shown in the figure, such transition delay can result in unexpected peak power or prolongs the response delay. The peak power might increase the risk of datacenter overload similar to power attack [38]. In addition, the long response delay might compromise the effectiveness of ANT-Man.



(a) *TrainTicket.*

(b) *SocialNetwork.*

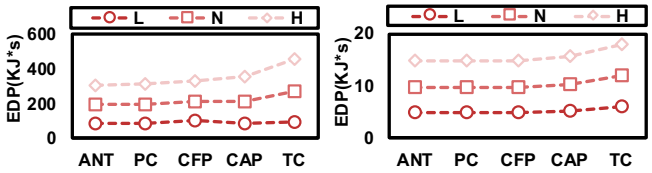
Figure 19: Variation of response delay with load level data.

C. The Extra Energy Saving of ANT-Man

We further investigate how much energy ANT-Man can save under different power over-subscription scenarios as well as load levels. In the experiment, we consider five power provisioning scenarios, namely 100% (PB), 90% (HPB), 85% (MPB) and 80% (LPB) of the nameplate power. In Figure 18, ANT-Man can save power for up to 40% in the PB scenario. It is significant since conventional power management approaches can hardly reduce power without any degradation on performance of the OLDI applications. Even when power budget decreases, ANT-Man can still achieve about 20% more power savings for *TrainTicket* and 10% for *SocialNetwork*.

D. Comparison with the Present Schemes

Both mean response delay and tail latency are important metrics for meeting the SLOs of applications. In Figure 19, we evaluate ANT-Man using these two metrics under different load levels. The mean response delay of ANT-Man (marked with ANT in the figures) is shorter than the others. Note that the delay of PC is similar to ANT if the load level is low.



(a) *TrainTicket.* (b) *SocialNetwork.*

Figure 20: The EDP of different schemes.

This is because PC slashes the power of the same group of microservices to reclaim the power budget. Nevertheless, with higher loading, the delay of traditional power management schemes increase rapidly since they allocate power without the awareness of IAV. They cannot take advantage of microservices whose execution is less sensitive to power reduction. In the rightmost figure, we compare the mean response delay and tail latency under different schemes at high load. It shows that TC has the worst SLOs while ANT-Man yields the best. For example, the 95^{th} tail latency of requests under ANT-Man is 9.7%-12.5% less than baseline TC. We also evaluate the energy delay production (EDP) of our design. EDP offers equal "weight" to either energy or performance degradation. In Figure 20, ANT-Man has the best performance-power trade-off while TC has the worst. Our results show that ANT-Man has 2%-40.1% of EDP lower than the others. Summarily, as compared to the conventional power management designs, ANT-Man makes better use of the limited power resources.

VII. Related Works

A. Power Management in Datacenters

In the last decades, datacenter power management has been examined to some depth [25], [35], [36], [38], [40], [49], [57], [58], [72]. Broadly, we classify existing techniques into three categories: aggressive off-peak provision, energy-proportional design, and user-specific optimization. Aggressive off-peak provision assumes that power demand surge does not occur very often [29], [36]. Thus datacenters could over-subscribe the power infrastructure and use power peak shaving in emergency [35], [36], [38], [49]. Energy-proportional design aims at improving server efficiency at low and moderate utilization [18], [54]. It is often possible to achieve higher energy proportionality by tuning various power modes [54], [55] or consolidating servers [53], [70]. Since many workloads are sensitive to the latency of power-saving mode [18], [27], user-specific power optimization frameworks have been proposed. These proposals incorporate end-to-end metrics and SLO targets from OLDI applications into power-saving decisions [38], [40], [54].

All the above designs could save considerable power, but they also introduce millisecond or second level control latency. In Table III, we categorize the state-of-the-art proposals into four classes based on control delay. Most of the power management frameworks on production systems such as Dynamo [72], battCapping [36] and multi-level shaving [61] have a relatively high latency of over 10 seconds. They focus more on optimizing the overall utilization of datacenters with less awareness of per-user or per-application requirements. Some application-specific optimizations like Rubik [54] still incur inactive-to-active delay caused by changing power-saving modes. Adrenaline [40] is one of the very-low-latency techniques which could pinpoint and boost certain tail queries at submicrosecond level. However, it requires hardware additions and it also needs several milliseconds to execute frequency transition.

Table III: The state-of-the-art power control techniques.

	Name	Delay	Service Specific	Hardware	Source
Fastest	**Adrenaline [40]**	≥10 ns	✔	Extra Circuit	UMich
Faster	**Adrenaline [40]**	us∼ms	✔	DVFS	UMich
	μDPM [25]	1ms	✔	Sleep State	UCR
	Caloree [59]	2 ms	✘	big.LITTLE	Chicago
	PowerNap [57]	1-10 ms	✘	Sleep State	UMich
	EEFL [37]	Few ms	✔	Duty Cycle	Rutgers
Fast	**Rubik [43]**	500 ms	✔	DVFS	MIT
	Pegasus [54]	5 s	✔	DVFS	Stanford
	Dynamo [72]	10 s	✘	DVFS	Facebook
	BattCap [36]	≥10.56s	✘	Battery	Microsoft
Slow	**Multi-level Shaving [61]**	Few min	✘	P-state	HP

There have been another line of work on managing renewable energy in green datacenters. For example, Blink [66] proposes to leverage fast power state transition to track renewable power variation. It is still coarse-grained power adaptation. Many prior studies use mechanical power switch [33], [48], [50] to select between different datacenter power supplies. The transfer time is short (typically at the microsecond level). However, these schemes are not suitable for management microservice energy consumption. Frequently switching workloads can also cause device ware-out issue. In addition, there are prior works focusing on the output speed of power supply [41], [52]. It may take several seconds to ramp power up as load changes. Since most renewable energy-powered datacenters use energy storage devices to handle power emergency [51], there is no need to perform super-fast load power tracking when renewable power supply changes. Although one can predict application characteristics to improve efficiency [34], it cannot provide highly granular power management in the microservice era.

B. System Researches on Microservices

Microservice is gaining popularity in cloud datacenters [3], [28], [42]. Most prior works emphasize on designing microservice applications [31], [79], or enhancing the robustness of this software architecture itself [28], [31], [49], [68]. A few proposals have focused on improving the performance of microservices [16], [30]. For example, Yu et al. focused on predicting QoS violations among massive microservices [30], [77]. Wenisch et al. conducted a series of researches on optimizing microsecond-scale services [69]. Neverthelees, very few works consider the power management of microservices.

Chou et al. [25] used prediction to bypass the OS-related transition delay. The proposed scheme named μDPM simply prolongs the execution of microservices without the consideration of their performance-power correlation. Meanwhile, it overlooks the macro control latency at the datacenter level. Microservice scheduling optimization might further facilitate power management and we will explore this in our future work.

Prior works have recognized the need for power control at a finer granularity. On the local server side, proposals [24], [56], [60] like Power Container [67] could assign the power to each running tasks on multi-cores. However, they mainly focus on how to model power for different tasks. Additionally, fine-grained power control techniques [44], [73], [76] have been proposed. on the datacenter side, per-application [38], [54] or per-request [40] power management may guarantee the SLO of OLDI workloads. For example, CFP [38] implements per-application power assignment in accordance with user preference. PoDD [78] is a hierarchical and distributed power management system for coupled workloads in the supercomputing domain. Still, these works are not agile enough for microservices no matter at the global level or local level.

VIII. Conclusions

The microservice architecture is redefining the cloud environment today. For datacenter architects, it opens up a new door for pushing the limit of system efficiency. In this paper, we show that such a great opportunity cannot be exploited unless the power management scheme takes into account microservice characteristics and becomes cloud-native. We propose ANT-Man, a novel framework that provides the necessary abstraction and optimization for highly efficient power management in the microservice environment. We show that such a agile, coordinated power management solution can yield a better power/performance trade-off. ANT-Man is applicable to many large-scale systems that requires granular power management under emerging applications. We expect that our work will provide valuable insights for both academics and practitioners in the design of the next-generation cloud infrastructure.

Acknowledgment

We thank all the reviewers for their valuable comments and feedbacks. This work is sponsored by the National Natural Science Foundation of China (NSFC No. 61972247). This work is also supported in part by NSF grants CCF 1822985 and CCF 1943490 (CAREER). Corresponding author is Chao Li from Shanghai Jiao Tong University.

References

[1] "Cloud native computing." [Online]. Available: https://www.cncf.io/

[2] "Intel 64 and ia-32 architectures software developer's manual," *Intel*, 2006. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-instruction-set-reference-manual-325383.pdf

[3] "Introduction to microservices," *Inc. Chris Richardson of Eventuate*, 2015. [Online]. Available: https://www.nginx.com/blog/introduction-to-microservices/

[4] "Introduction to dubbo," *Baeldung*, 2018. [Online]. Available: https://www.baeldung.com/dubbo

[5] "Alibaba cluster data," 2019. [Online]. Available: https://github.com/alibaba/clusterdata

[6] "Central limit theorem," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Central_limit_theorem

[7] "Grpc: A high performance open-source universal rpc framework," 2019. [Online]. Available: https://grpc.io/

[8] "Introduction to cache allocation technology in the intel xeon processor e5 v4 family," 2019. [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-cache-allocation-technology

[9] "Jaeger," 2019. [Online]. Available: https://www.jaegertracing.io/

[10] "Kubernetes," 2019. [Online]. Available: https://kubernetes.io/

[11] "Memchached," 2019. [Online]. Available: https://hub.docker.com/_/memcached

[12] "Microservices on aws," *Amazon Web Services*, 2019. [Online]. Available: https://docs.aws.amazon.com/aws-technical-content/latest/microservices-on-aws/microservices-on-aws.pdf

[13] "Swarm: A docker-native clustering system," 2019. [Online]. Available: https://github.com/docker/swarm

[14] "Xen credit scheduler," 2019. [Online]. Available: https://wiki.xen.org/wiki/Credit_Scheduler

[15] "Xen power management," 2019. [Online]. Available: https://wiki.xenproject.org/wiki/Xen_power_management

[16] M. Amaral, J. Polo, D. Carrera, I. Mohomed, M. Unuvar, and M. Steinder, "Performance evaluation of microservices architectures using containers," in *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications (NCA)*, 2015.

[17] L. Barroso, M. Marty, D. Patterson, and P. Ranganathan, "Attack of the killer microseconds," in *ACM Communication*, 2017.

[18] L. Barrosoet, U. Hölzle, and P. Ranganathan, "The datacenter as a computer: An introduction to the design of warehouse-scale machines," in *Synthesis Lectures on Computer Architecture*, 2018.

[19] M. Becker and S. Chakraborty, "Measuring software performance on linux," in *ArXiv*, 2018.

[20] A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar, "The need for speed and stability in data center power capping," in *Proceedings of the International Green Computing Conference (IGCC)*, 2012.

[21] A. Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology," in *ArXiv*, 2018.

[22] J. Brutlag, H. Hutchinson, and M. Stone, "User preference and search engine latency," 2008.

[23] A. Buck, "Microservices architecture style," 2019. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/guide/architecture-styles/microservices

[24] I. Canturk and M. Margaret, "Phase characterization for power: Evaluating control-flow-based and event-counter-based techniques," in *Proceedings of the 12th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2006.

[25] C. Chou, L. Bhuyan, and D. Wong, "μdpm: Dynamic power management for the microsecond era," in *Proceedings of the 25th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2019.

[26] T. Cormen, "Introduction to algorithms," in *MIT Press*, 2009.

[27] J. Dean and L. Barroso, "The tail at scale," in *ACM Communication*, 2013.

[28] D. Escobar, D. Cardenas, R. Amarillo, E. Castro, K. Garcés, C. Parra, and R. Casallas, "Towards the understanding and evolution of monolithic applications as microservices," in *Proceedings of the 42th XLII Latin American Computing Conference (CLEI)*, 2016.

[29] X. Fan, W. Weber, and L. Barroso, "Power provisioning for a warehouse-sized computer," in *Proceedings of the 34th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2007.

[30] Y. Gan, M. Pancholi, D. Cheng, S. Hu, Y. He, and C. Delimitrou, "Seer: Leveraging big data to navigate the complexity of cloud debugging," in *Proceedings of the 10th USENIX Conference on Hot Topics in Cloud Computing (HotCloud)*, 2018.

[31] Y. Gan, Y. Zhang, D. Cheng, A. Shetty, P. Rathi, N. Katarki, and K. Hu, "An open-source benchmark suite for microservices and their hardware-software implications for cloud & edge systems," in *Proceedings of the 24th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[32] J. Garrison and K. Nova, "Cloud native infrastructure: Patterns for scalable infrastructure and applications in a dynamic environment," in *O'Reilly Media*, 2017.

[33] I. Goiri, W. Katsak, K. Le, T. D. Nguyen, and R. Bianchini, "Parasol and greenswitch: Managing datacenters powered by renewable energy," in *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '13, 2013.

[34] I. Goiri, K. Le, M. E. Haque, R. Beauchea, T. D. Nguyen, J. Guitart, J. Torres, and R. Bianchini, "Greenslot: Scheduling energy consumption in green datacenters," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11, 2011.

[35] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar, "Benefits and limitations of tapping into stored energy for datacenters," in *Proceedings of the 38th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2011.

[36] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar, "Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2012.

[37] M. Haque, Y. He, S. Elnikety, T. Nguyen, R. Bianchini, and K. McKinley, "Exploiting heterogeneity for tail latency and energy efficiency," in *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2017.

[38] X. Hou, L. Hao, C. Li, Q. Chen, W. Zheng, and M. Guo, "Power grab in aggressively provisioned data centers: What is the risk and what can be done about it," in *Proceedings of the 36th IEEE International Conference on Computer Design (ICCD)*, 2018.

[39] X. Hou, J. Liu, C. Li, and M. Guo, "Unleashing the scalability potential of power-constrained data center in the microservice era," in *Proceedings of the 48th International Conference on Parallel Processing (ICPP)*, 2019.

[40] C. Hsu, Y. Zhang, M. Laurenzano, D. Meisner, T. Wenisch, J. Mars, and R. Dreslinski, "Adrenaline: Pinpointing and reining in tail queries with quick voltage boosting," in *Proceedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2015.

[41] Y. Hua, C. Li, W. Tang, L. Jiang, and X. Liang, "Building fuel powered supercomputing data center at low cost," in *Proceedings of the 29th ACM on International Conference on Supercomputing*, ser. ICS '15, 2015, p. 241–250.

[42] G. Kakivaya, L. Xun, R. Hasha, S. Ahsan, T. Pfleiger, R. Sinha, and M. Mohsin, "Service fabric: A distributed platform for building microservices in the cloud," in *Proceedings of the 13th EuroSys Conference (EuroSys)*, 2018.

[43] H. Kasture, D. Bartolini, N. Beckmann, and D. Sanchez, "Rubik: Fast analytical power management for latency-critical systems," in *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2015.

[44] W. Kim, M. Gupta, G. Wei, and D. Brooks, "System level analysis of fast, per-core dvfs using on-chip switching regulators," in *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2008.

[45] R. O. L. Breiman, J.H. Friedman and C. Stone, "Classification and regression t rees (monterey, ca: Wadsworth," in *BreimanClassification and Regression Trees*, 1984.

[46] J. Lewis and M. Fowler, "Microservices a definition of this new architectural term," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html

[47] C. Li, C. Ding, and K. Shen, "Quantifying the cost of context switch," in *Experimental Computer Science*, 2007.

[48] C. Li, Y. Hu, J. Gu, J. Yuan, and T. Li, "Oasis: Scaling out datacenter sustainably and economically," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 7, pp. 1960–1973, 2017.

[49] C. Li, Z. Wang, X. Hou, H. Chen, X. Liang, and M. Guo, "Power attack defense: Securing battery-backed data centers," in *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[50] C. Li, A. Qouneh, and T. Li, "Iswitch: Coordinating and optimizing renewable energy powered server clusters," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12, 2012.

[51] C. Li, R. Wang, D. Qian, and T. Li, "Managing server clusters on renewable energy mix," *ACM Transactions on Autonomous and Adaptive Systems*, vol. 11, no. 1, 2016.

[52] C. Li, R. Zhou, and T. Li, "Enabling distributed generation powered sustainable high-performance data center," in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, ser. HPCA '13, 2013, p. 35–46.

[53] M. Lin, A. Wierman, L. Andrew, and E. Thereska, "Dynamic right-sizing for power-proportional data centers," in *IEEE/ACM Transactions on Networking (TON)*, 2013.

[54] D. Lo, L. Cheng, R. Govindaraju, L. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceedings of the 41st ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2014.

[55] D. Lo and C. Kozyrakis, "Dynamic management of turbomode in modern multi-core chips," in *Proceedings of the 20th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2014.

[56] J. McCullough, Y. Agarwal, J. Chandrashekar, S. Kuppuswamy, A. Snoeren, and R. Gupta, "Evaluating the effectiveness of model-based power characterization," in *Proceedings of the USENIX Conference on USENIX Annual Technical Conference (ATC)*, 2011.

[57] D. Meisner, B. Gold, and T. Wenisch, "Powernap: eliminating server idle power," in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2009.

[58] D. Meisner, C. Sadler, L. Barroso, W. Weber, and T. Wenisch, "Power management of online data-intensive services," in *Proceedings of the 38th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2011.

[59] N. Mishra, C. Imes, J. Lafferty, and H. Hoffmann, "Caloree: Learning control for predictable latency and low energy," in *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2018.

[60] R. Monfort, M. González, X. Martorell, N. Navarro, and E. Ayguadé, "Decomposable and responsive power models for multicore processors using performance counters," in *Proceedings of the 24th ACM International Conference on Supercomputing (ICS)*, 2010.

[61] R. Raghavendra, P. Ranganathan, V. Talwar, Z. Wang, and X. Zhu, "No "power" struggles: Coordinated multi-level power management for the data center," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[62] M. Richards, "Microservice vs service-oriented architecture," in *O'Reilly Media*, 2015.

[63] A. Roy, H. Zeng, J. Bagga, G. Porter, and A. Snoeren, "Inside the social network's (datacenter) network," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication(SIGCOMM)*, 2015.

[64] V. Sakalkar, V. Kontorinis, D. Landhuis, S. Li, D. De Ronde, T. Blooming, A. Ramesh, J. Kennedy, C. Malone, J. Clidaras, and P. Ranganathan, "Data center power oversubscription with a medium voltage power plane and priority-aware capping," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20, New York, NY, USA, 2020, p. 497–511.

[65] S.Frey, C. Lüthje, R. Teckelmann, and C. Reich, "Adaptable service level objective agreement (a-slo-a) for cloud services," in *Proceeding of the International Conference on Cloud Computing and Services Science (CLOSER)*, 2013.

[66] N. Sharma, S. Barker, D. Irwin, and P. Shenoy, "Blink: Managing server clusters on intermittent power," in *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVI. New York, NY, USA: Association for Computing Machinery, 2011.

[67] K. Shen, A. Shriraman, S. Dwarkadas, X. Zhang, and Z. Chen, "Power containers: An os facility for fine-grained power and energy management on multicore servers," in *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[68] A. Sriraman and T. Wenisch, "μsuite: A benchmark suite for microservices," in *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, 2018.

[69] A. Sriraman and T. Wenisch, "μtune: Auto-tuned threading for oldi microservices," in *Proceedings of the 13th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[70] D. Wong and M. Annavaram, "Knightshift: Scaling the energy proportionality wall through server-level heterogeneity," in *Proceedings of the*

45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO), 2012.

[71] M. Wu, "Taking the cloud-native approach with microservices," 2017. [Online]. Available: https://cloud.google.com/files/Cloud-native-approach-with-microservices.pdf

[72] Q. Wu, Q. Deng, L. Ganesh, C. Hsu, Y. Jin, S. Kumar, and Y. Song, "Dynamo: Facebook's data center-wide power management system," in *Proceedings of the 43rd ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2016.

[73] G. Yan, Y. Li, Y. Han, X. Li, M. Guo, and X. Liang, "Agileregulator: A hybrid voltage regulator scheme redeeming dark silicon for power efficiency in a multicore architecture," in *Proceedings of the 18th IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2012.

[74] H. Yang, Q. Chen, M. Riaz, Z. Luan, L. Tang, and J. Mars, "Powerchief: Intelligent power allocation for multi-stage applications to improve responsiveness on power constrained cmp," in *Proceedings of the 44th ACM/IEEE Annual International Symposium on Computer Architecture (ISCA)*, 2017.

[75] H. Ye, "Using cpufreq on linux servers to manage power consumption," in *Lenove Press*, 2018.

[76] G. Yu and C. Delimitrou, "Power management of datacenter workloads using per-core power gating," in *IEEE Computer Architecture Letters (CAL)*, 2009.

[77] G. Yu and C. Delimitrou, "The architectural implications of cloud microservices," in *IEEE Computer Architecture Letters (CAL)*, 2018.

[78] H. Zhang and H. Hoffmann, "Podd: power-capping dependent distributed applications," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, 2019.

[79] X. Zhou, X. Peng, T. Xie, J. Sun, C. Xu, C. Ji, and W. Zhao, "Poster: Benchmarking microservice systems for software engineering research," in *Proceedings of the IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, 2018.