Increasing the Profit of Cloud Providers through DRAM Operation at Reduced Margins

Christos Kalogirou, Christos D. Antonopoulos,
Nikolaos Bellas and Spyros Lalis
University of Thessaly
Volos, Greece
{hrkalogi,cda,nbellas,lalis}@uth.gr

Lev Mukhanov and Georgios Karakonstantis
Queen's University
Belfast, United Kingdom
{L.Mukhanov,G.Karakonstantis}@qub.ac.uk

***Abstract*—Energy reduction is a key objective in cloud computing, and DRAM memories are responsible for an important amount of the energy consumption of data center nodes. Vendors adopt very conservative margins for DRAM operating parameters, such as the refresh rate and supply voltage, to guarantee correct operation even under the worst process variation and operating conditions. In this paper, we investigate the exploitation of DRAM margins to improve the energy efficiency of data center nodes, without triggering penalties due to service level agreement (SLA) violations. We introduce a model that captures the most important aspects of job management and system configuration. We also introduce *RM-DRAM*, a scheduling and node configuration policy that exploits the extended margins of DRAMs to reduce the operator's cost, considering the trade-off between the cost of energy consumption and potential SLA violations. *RM-DRAM* also employs cost-aware (rather than threshold-based) VM consolidation. We extract the parameters used in the simulation (particularly power consumption and error rates) by characterizing a commercial ARM-based server. We perform simulations to evaluate the effectiveness of our approach, showing that significant gains, up to 34.84% and 29.53% in energy and cost, respectively, can be achieved compared with a state-of-the-art policy.**

***Index Terms*—DRAM reduced margins, energy efficiency, scheduling, cost effectiveness**

I. Introduction

Data centers account for 2-3% of the worldwide energy consumption and the main contributor to this energy consumption are enterprise servers which represent around 60% of the overall data center energy [1]. Recent studies show that Dynamic Random Access Memory (DRAM) is rapidly becoming a major power consumer on servers, already contributing a significant fraction (around 30%) of the total power consumption of the node [2], [3]. As the memory footprint of generated data increases, raising the pressure for higher DRAM densities, it is expected that the power consumed by DRAMs will keep growing, requiring immediate attention [4].

However, scaling down the DRAM energy is extremely challenging, due to the adoption by manufacturers of conservative voltage margins and frequent refresh cycles to address the increased variability of cell retention time within, and across DRAM chips [5]. To address the inherent limited cell retention time, modern DRAMs employ an *Auto-Refresh* mechanism that periodically recharges all cells at a specific *refresh period* (RP). Conventionally, the refresh period is determined based on the worst-case retention time across all manufactured cells, which is estimated assuming extremely rare, worst-case combinations of operating conditions (e.g. extremely high temperature and worst-case access and stored data patterns), also including an additional non-negligible safety guardband [6]. Based on that philosophy, DDRx technologies adopt the same refresh period (i.e 64 ms, or 32 ms in case of high temperature) even if, in reality, most cells may have much higher retention time. Such a pessimistic refresh period leads to considerable power and performance overheads, which are expected to worsen as DRAM densities increase. It has been projected that future 64 Gb DRAM chips will spend 35% of the memory power and 25% of the available DRAM chip bandwidth on refreshes only [7]. Similarly to the refresh period, the nominal supply voltage (V_{DD}), of DRAMs is chosen conservatively, based on pessimistic assumptions, further hampering the DRAM energy scaling.

This reality has turned the attention on schemes that trade-off DRAM energy with reliability by either adopting high refresh periods only for few cells [7], [8] or by exploiting the inherent error resiliency of various applications to find phases that can relax the refresh period [9]–[12].

Even though many works have studied DRAM reliability, to the best of our knowledge, none of them systematically investigated reliability, energy and cost gains in data centers by operating DRAMs under relaxed parameters. In fact previous work on the reduction of energy consumption of data centers has focused only on CPUs, mainly targeting better consolidation of Virtual Machines (VMs) [13] and exploitation of Dynamic Voltage and Frequency Scaling (DVFS) [14].

In this paper, we study the trade-off between the reduced energy cost due to operation at extended margins and the SLA violation penalties due to the increased probability of DRAM errors. Compared with [13], the results from our experiments indicate 34.84% energy gains at the plug in nodes with 512 GB DRAM at 50°C. Out of this reduction, 16.69% is due to the relaxation of the DRAM refresh period, and the rest due to more effective VM consolidation. This energy reduction translates to 29.53% reduced costs for the data center operator. Moreover, energy and cost gains are higher in nodes with larger memory capacity, which shows that our approach will scale favorably in future data-dominated data centers. The

978-1-7281-6095-5/20/$31.00 ©2020 IEEE
DOI 10.1109/CCGrid49817.2020.00-38

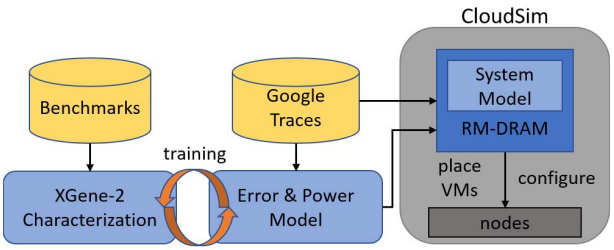

Fig. 1: Overview of the proposed approach.

contributions of this paper are the following:

- We introduce an analytic and architecture-agnostic model, which considers the trade-off between the reduction of energy consumption due to extended margins configuration of DRAMs and the risk of potential SLA violations.
- Based on the model, we introduce Reduced-Margins DRAM (*RM-DRAM*), a novel, cost-driven scheduling and node configuration policy for cloud data centers. This policy considers DRAM configuration at extended operating points, on top of cost-aware VM consolidation.
- We evaluate our approach in CloudSim [15] using Google traces [16].
- We derive a realistic node profile to drive the simulations by characterizing and modeling the workload aware reliability and power effects of the DRAM operation at extended points on an ARM-based, 64-bit Ampere Computing X-Gene2 server system. The model uses features available in Google traces (such as CPU utilization, memory footprint, and memory access rate). ARM-based servers lately attract significant attention due to their power efficiency [17], thus our work is timely.

Figure 1 illustrates the main elements of our approach.

The rest of the paper is organized as follows: In Sections II and III we introduce the system model, and the *RM-DRAM* policy, respectively. In Section IV we discuss the methodology and results of the characterization of DRAM error and power behavior on the X-Gene2 system. Section V discusses the experimental evaluation and comparison of *RM-DRAM* with a data center scheduling policy focused on energy optimization [13]. Section VI provides an overview of related work. Finally, Section VII concludes the paper.

II. Problem Modeling

This section introduces an analytic model for the estimation of the data center operating cost with DRAM operation at reduced margins. It captures key server configuration and VM scheduling aspects, while making realistic assumptions that simplify the problem formulation without loss of generality.

A. Nodes and DRAM operating points

We assume a data center with N server nodes $n_i, 1 \leq i \leq N$. All nodes have the same CPU and DRAM capacity, denoted by CPU and Mem, respectively. The DRAM of each node n_i operates at refresh period RP_i. By default, this is set to a predefined nominal value, which is the same for all nodes.

It is possible to increase the DRAM refresh period beyond the nominal value to exploit the available cell retention margins. This can reduce power consumption, at the risk of introducing errors to the system. The impact of increased refresh periods in terms of system power consumption and errors is discussed in Section IV.

B. Workload – VMs

Each job is packaged as a VM (the notion of a VM serves as an abstraction for isolated/contained execution). Let there be M different VMs $VM_m, 1 \leq m \leq M$, which are submitted for execution in the data center at various points in time. Each VM has different peak resource requirements (we focus on CPU and memory), declared in a service level agreement (SLA) to be respected by the provider.

Let Mem_m^{SLA} be the peak memory requirement of VM_m specified in the SLA. We assume that the requested memory is allocated in full before a VM starts running and remains allocated to the VM during its entire execution. While hypervisors typically apply transparent page sharing to a limited extent in order to facilitate memory overcommit, in this model, we assume that memory is not shared among VMs. This assumption simplifies resource management and error accounting, without affecting the validity of the model.

Similarly, let CPU_m^{SLA} be the upper bound for the CPU capacity that will be requested by VM_m, also defined in the SLA. This is expressed as the percentage of the full CPU capacity of the node. CPU requirements are, however, characterized by high volatility. Let $CPU_m^{req} \leq CPU_m^{SLA}$ be the CPU capacity that is requested by VM_m at runtime, again expressed as a fraction of the full CPU capacity.

C. VM scheduling

The scheduling/placement of VMs on nodes is performed periodically. Each period p has a fixed length, divided to K timeslots of $slotT$ time units each. For simplicity, we assume that new VMs are accepted and existing VMs terminate at the boundaries of scheduling periods. We use a binary matrix L to encode whether VM_m is live during period p (VM_m was submitted for execution before p and has not yet terminated). If VM_m is live then $L[m,p] = 1$, otherwise $L[m,p] = 0$.

Between periods, the scheduler decides and applies (a) the placement of new VMs and rescheduling of existing VMs to other nodes, and (b) the refresh period $RP_{i,p}$ for the DRAMs of each node n_i for the next period p.

The assignment of VMs to nodes for the period p is encoded in the binary hosting matrix H, where $H[i,m,p] = 1$ if n_i hosts VM_m during p, else $H[i,m,p] = 0$. A node n_i must have enough memory to accommodate the VMs assigned to it: $Mem_i \geq \sum_{m=1}^{M} H[i,m,p] \times Mem_m^{SLA}$. We assume that the number of nodes is sufficient to accommodate all live VMs at any point in time (data center has ample capacity) and that every live VM is placed on some node, in other words $L[m,p] = 1 \Leftrightarrow \exists n_i : H[i,m,p] = 1$. Given that each

VM is assigned to a single node $\sum_{i=1}^{N} H[i,m,p] = 1 \ \forall m : L[m,p] = 1$.

Apart from placing new VMs on nodes, the scheduler may decide to reallocate existing VMs to different nodes. We assume that the applications running in the VMs are stateless. Thus, VMs can be simply restarted on a different node, without needing to record, transfer and restore state information. We assume the startup time Res_m of a restarted VM_m to be a function of its memory footprint Mem_m^{SLA} and that $1 \leq Res_m \ll K$, i.e., startup takes at least one timeslot but is much shorter than the scheduling period.

We use matrix X to capture the VM reallocation and restart overhead, where $X[i,m,p,k] = 1$ if VM_m actually runs on n_i in the kth timeslot of period p, otherwise $X[i,m,p,k] = 0$. If VM_m is scheduled for reallocation to n_i in p, then $X[i,m,p,k] = 0$, $0 \leq k \leq Res_m - 1$ where VM_m is still in its startup phase, while $X[i,m,p,k] = 1$, $Res_m \leq k < K$.

Finally, we define the workload of node n_i during period p as $W_{i,p} = \{VM_m | H[i,m,p] = 1\}$. This is the set of all VMs that are scheduled for execution on the node for that period. The workload may change between periods, as follows

$$W_{i,p} = W_{i,p-1} - W_{i,p}^{left} + W_{i,p}^{new} \tag{1}$$

where $W_{i,p}^{left}$ is the set of VMs that have terminated in the previous period or are selected to be reallocated to another node, and $W_{i,p}^{new}$ are the VMs placed on the node in this period due to new VM arrivals or rescheduling of existing VMs.

D. Errors due to relaxed refresh rate of DRAMs

Server DRAMs are equipped with error checking and correction (ECC) circuitry. Depending on the algorithm used, ECCs can detect-and-correct up to a certain number of bit errors (detected correctable errors - CEs) and detect-only up to an additional number of errors per memory word (detected uncorrectable errors - UEs). In the following, we explain how we incorporate the effects of CEs and UEs in our model.

1) Correctable errors (correction/logging overhead): CEs are not fatal for node operation and VM execution. Still, their logging and correction take away a portion of the CPU capacity, which effectively increases the overall system load. We calculate the CPU capacity required for the logging and correction of CEs that occur in n_i during p as

$$CPU_{i,p}^{CE} = \frac{overhead}{K \times slotT} \times CE_{i,p} \times CPU^{CE} \tag{2}$$

where $overhead$ and CPU^{CE} are the required amount of time and CPU capacity for reporting and respectively correcting a CE, and $CE_{i,p}$ is the total number of CEs that occur in n_i during p. In turn, we let

$$CE_{i,p} = \sum_{m=1}^{M} H[i,m,p] \times CE_{i,m,p} \tag{3}$$

where $CE_{i,m,p}$ is the expected number of CEs in VM_m during p on n_i when the DRAM refresh period is set to RP_i. Since VMs do not share memory, it is reasonable to assume that CEs occur in different VMs independently.

2) Uncorrectable errors (node crash failures): UEs can be detected, but they cannot be corrected. The default behavior in most popular operating systems, such as Linux, is a kernel panic. Hence we assume that UEs lead to a node crash.

We let $P_{i,m,p}^{UE}$ represent the probability of VM_m suffering an UE when running on node n_i during period p. A node will crash as soon as one of the hosted VMs suffers an UE. Given that VMs do not share memory, any VM is a candidate to independently suffer an UE and a single UE is sufficient to crash the node, the probability of n_i failing in period p equals

$$Pfail_{RP_{i,p}} = P\left(\bigcup_{VM_m \in W_{i,p}} UE_{i,m,p}\right) \tag{4}$$

Taking a pessimistic approach, we assume that such a node crash always occurs at the beginning of a period and that all VMs that are hosted on the crashed node remain unavailable during the entire period. At the same time, we assume that the VMs of a failed node can be reallocated and restarted in the next period on another node without further delay.

E. CPU load and CPU capacity allocation to VMs

The CPU load of a node in a given timeslot is equal to the total effective requested CPU capacity of all VMs running on it. Moreover, the load increases if CEs occur at the node. Thus the total load of n_i in timeslot k or period p is estimated as

$$Load_{i,k} = \sum_{m=1}^{M} (X[i,m,p,k] \times CPU_{m,p}^{req}) + CPU_{i,p}^{CE} \tag{5}$$

In turn, the CPU capacity allocated to each VM in a given timeslot depends on whether the node is under- or over-loaded:

$$CPU_{m,k}^{alloc} = \begin{cases} X[i,m,p,k] \times CPU_{m,p}^{req} & \text{if } Load_{i,k} \leq 1 \\ X[i,m,p,k] \times (1 - CPU_{i,p}^{CE}) & \\ \times \frac{CPU_{m,p}^{req}}{Load_{i,k} - CPU_{i,p}^{CE}} & \text{if } Load_{i,k} > 1 \end{cases} \tag{6}$$

If a node is under-loaded, every VM running on it will get the requested CPU capacity. Otherwise, the available CPU capacity, after taking into account the handling of CEs, is shared among VMs proportionally to their requests. Here we assume equal VM priorities, but it is straightforward to adapt the model to a priority-based CPU allocation scheme.

F. SLA violation cost

A VM may not get the requested CPU capacity at every timeslot. In these cases, the provider pays an SLA violation penalty to the VM owner.

The SLA violation cost is modeled similarly to [18]. Concretely, we assume it is equal to the product of the normalized extent of the violation (expressed as the difference between the provisioned CPU capacity CPU_m^{SLA} and the one that was actually allocated $CPU_{m,k}^{alloc}$, divided by CPU_m^{SLA}), and the duration of the violation. For any given timeslot k, this is

$$SLAV_{i,m,k} = q_m \times \frac{CPU_m^{SLA} - CPU_{m,k}^{alloc}}{CPU_m^{SLA}} \times SlotT \times Price_m \tag{7}$$

where q_m is a multiplier that increases the severity of the penalty according to the importance of VM_m, and $Price_m$ is the price charged to the client for executing VM_m for a time unit, as both agreed in the SLA. Equation 7 applies to all sources of SLA violations for VM_m: reallocations to another node, node overloads and node failures.

Based on the above, the SLA violation cost over an entire period for all VMs hosted on a node n_i that does not fail, is

$$SLAV_{i,p}^{nofail} = \sum_{k=1}^{K} \sum_{m=1}^{M} X[i,m,p,k] \times SLAV_{i,m,k} \quad (8)$$

If n_i fails, none of the VMs will get any CPU capacity, in which case the total SLA violation cost for period p is

$$SLAV_{i,p}^{fail} = K \times SlotT \times \sum_{m=1}^{M} H[i,m,p] \times q_m \times Price_m \quad (9)$$

G. Energy costs

Let $Power(RP_{i,p}, u)$ be the power consumed by node n_i with a DRAM refresh period $RP_{i,p}$ and CPU utilization u. In turn, the CPU utilization of n_i in timeslot k is given by $u_{i,k} = \min\{Load_{i,k}, 1\}$. Then, assuming a fixed $UnitPrice$ for energy, the total energy cost for n_i over period p is

$$Energy_{i,p} = UnitPrice \times \sum_{k=1}^{K} Power(RP_{i,p}, u_{i,k}) \times SlotT \quad (10)$$

Note that if n_i is not assigned any VMs in p, it will remain idle and its utilization u for that period will be 0. In our experiments, we assume that idle nodes are shut-down. In the same spirit, we assume that failed nodes do not consume any energy during the entire period where the crash failure occurs.

H. Operating cost

The cost for operating node n_i during period p, assuming it has been assigned workload $W_{i,p}$ and its DRAM is configured to operate with refresh period RP_i, is estimated by

$$Cost_{i,p}^{W_{i,p}} = Pfail_{RP_{i,p}} \times SLAV_{i,p}^{fail} + (1 - Pfail_{RP_{i,p}}) \times (SLAV_{i,p}^{nofail} + Energy_{i,p}) \quad (11)$$

More specifically, there are two cases (multiplied by the respective probability). The node may fail due to an UE, in which case the provider will pay the SLA violation cost for all the VMs on the node for the entire period. If the node does not experience an UE (and thus does not fail), the provider may still pay SLA violation costs due to VM reallocation and restart, and/or node overload, plus the cost of the energy consumed by the node to run the VMs at the configured DRAM refresh period.

III. Scheduling and System Configuration Policy

Ideally, a scheduling algorithm would find the optimal VM placements and DRAM refresh periods for all nodes and for each period, so as to minimize the total operating cost of a data center over all periods. One problem is that, in practice, VM workloads and CPU requirements are not known in advance. Moreover, even if these were known, this optimization problem is NP-hard even for a single period.

In this section, we first introduce some approximations in order to (i) make the problem tractable, and (ii) do so with realistic input. Then, we introduce a workload scheduling and node configuration policy which focuses on maximizing the profit margin for data center infrastructure providers by exploiting DRAM operation at reduced margins.

A. Approximations

In a realistic setting, the future behavior of VMs is not known in advance. As a prediction of VM CPU requirements during the next period, we use a weighted average of CPU occupancy by each VM during previous scheduling periods with exponentially decreasing weights for older observations.

Moreover, we focus on greedily maximizing operator profit at the granularity of each scheduling period, based on Equation 11. In a similar vein, before each period, a cost-driven approach is applied to identify the preferred configuration of each node individually rather than at the data center-level. This approach improves the scalability of the scheduling policy with respect to the total number of VMs and nodes. The only step requiring a global overview of node status at the data center is the allocation of newly arriving and reallocated VMs to nodes, however, this typically involves just a few VMs.

B. VM Scheduling and Node Configuration – RM-DRAM

We introduce ReducedMargins-DRAM (*RM-DRAM*), a VM scheduling and node configuration policy that tries to reduce the cost for the cloud infrastructure provider by exploiting the refresh margins of DRAMs in an educated manner. In a nutshell, *RM-DRAM* exploits the extended margins of DRAMs on top of VM consolidation. For VM consolidation, we follow the approach in [13], however we propose cost-driven heuristics. To the best of our knowledge, we are the first to propose a policy that considers DRAM operation at reduced margins on top of VM consolidation.

RM-DRAM is executed at the boundaries of scheduling periods and identifies a good balance between energy cost reduction and the cost of potential SLA violation penalties, by applying a set of heuristics to select a mapping of VMs to nodes and decide the appropriate DRAM refresh period for each node. More specifically, for each scheduling period p, the policy executes a series of steps, described below.

During the first step, equation 11 is used to calculate the *a posteriori* optimal RP_i configuration (nominal or extended) for each node n_i independently, according to the behavior of VMs on that node in the previous period.

For this configuration, node overloads are identified according to the past behavior of VMs and VMs are selected to be

allocated to other nodes. Algorithm 1 describes this step in detail. We use the information of past CPU utilization (with exponential aging) to predict CPU requirements of each VM during the next period. A node is considered to be overloaded if the expected CPU utilization of VMs allocated on that node exceeds a CPU utilization threshold (thr_{util}). Also, overloaded nodes are checked – in parallel – to select the VMs that need to be rescheduled, in a cost-driven manner. A VM is selected to be reallocated to another node and restarted only if this is expected to introduce less cost than risking to leave the node in an overloaded state. The policy selects for reallocation the VM that will introduce the lowest cost (lines 9-14). Finally, VMs selected for reallocation are added to the set of VMs that need to be rescheduled ($VMsForScheduling_p$), and the workload is updated accordingly. This is repeated until no VM reallocation is deemed cost-effective (lines 15-20) or the node in question is no longer considered to be overloaded (line 6).

In the third step, the policy allocates to nodes the VMs that have been selected for reallocation and newly arriving VMs. We sort the VMs in the $VMsForScheduling_p$ set in decreasing CPU utilization order to allocate them to nodes with a Knapsack-like approach. We estimate the effect of allocating the VM at hand on the cost of each node n_i (energy cost and SLA violations cost), assuming the node will maintain the *a posteriori* optimal RP_i calculated during the first step. Each VM in the $VMsForScheduling_p$ is eventually hosted on the node that introduces the lowest cost increment.

At this point, the algorithm identifies opportunities to power-off nodes. Starting from the least utilized node, it checks whether all hosted VMs can be allocated to other nodes without overloading them. The destination node for this reallocation is chosen using the heuristic discussed previously.

The final step, described in Algorithm 2, is to find the optimal configuration independently for each node, given the VM placement for the next scheduling period and the predictions for future CPU requirements of each VM. The optimal node configuration minimizes the cost at the node level, therefore for a fixed VM-to-nodes mapping the total cost for the entire data center as well, taking into account both energy consumption and potential SLA violation costs. Again, we use Equation 11 to estimate the cost for each node/memory configuration, and apply the one that introduces the lowest cost. This step is similar to the first step. In the first step, however, we calculate the *a posteriori* optimal configuration of each node, given the known behavior of VMs during the previous period. In this step, we select the operating point for the next period, given the (potentially different, due to reallocation and assignment of new VMs) workload of the node for that period, and estimating future CPU requirements of the workload using past information.

Algorithm 1 Overload detection and VM selection for reallocation

Input: n_i, $W_{i,p}$
Output: $W_{i,p}^{left}$, $VMsForScheduling_p$

1: estimate CPU utilization u_i
2: **if** $u_i > thr_{util}$ **then**
3: **for** VM_m in $W_{i,p}$ **do**
4: $Cost_m^{realloc} \leftarrow \sum_{k=1}^{K} SLAV_{i,m,k}^{realloc}$
5: **end for**
6: **while** $u_i > thr_{util}$ **do**
7: $Cost_{min} \leftarrow Cost_i^{W_{i,p}}$
8: $VM^{realloc} \leftarrow NULL$
9: **for** VM_m in $W_{i,p}$ **do**
10: **if** $Cost_m^{realloc} + Cost_i^{W_{i,p}-\{VM_m\}} < Cost_{min}$ **then**
11: $Cost_{min} = Cost_m^{realloc} + Cost_i^{W_{i,p}-\{VM_m\}}$
12: $VM^{realloc} \leftarrow VM_m$
13: **end if**
14: **end for**
15: **if** $VM^{realloc}! = NULL$ **then**
16: $W_{i,p}^{left}$ add $\{VM^{realloc}\}$
17: $VMsForScheduling_p$ add $\{VM^{realloc}\}$
18: **else**
19: ***break***
20: **end if**
21: re-evaluate u_i
22: **end while**
23: **end if**

Algorithm 2 Node configuration for cost minimization

Input: n_i, $RPList$
Output: Node n_i configuration at $RP_i^{efficient}$

1: $Cost_{min} \leftarrow \infty$
2: $RP_i^{efficient} \leftarrow RP_i^{nominal}$
3: **for** RP_i in $RPList$ **do**
4: estimate $Cost_i^{W_{p,i}}$ at RP_i
5: **if** $Cost_{min} > Cost_i^{W_{p,i}}$ **then**
6: $Cost_{min} \leftarrow Cost_i^{W_{p,i}}$
7: $RP_i^{efficient} \leftarrow RP_i$
8: **end if**
9: **end for**
10: configure node n_i at $RP_i^{efficient}$

IV. Characterization and Modeling of DRAM Power and Error Behavior

This section presents the details of the DRAM error and power models that were developed for estimating $CE_{i,m,p}$, $P_{i,m,p}^{UE}$ and $Power(RP_{i,p}, u_{i,k})$ under various refresh periods in Equations 3, 4 and 10, respectively.

A. DRAM error modeling

We build a DRAM error model that considers not only the DRAM temperature and the refresh period, as prior works [7], [19], but also workload specific features. Such a model allows us to distinguish the DRAM reliability behavior based on the different traces that can be simulated in CloudSim (and which provide information about access rates and memory footprint),

while considering recent findings that show that DRAM errors are workload-dependent and may vary by 8x across workloads [20]. Formally, our model predicts a target DRAM error metric (i.e. $CE_{i,m,p}$ or $P^{UE}_{i,m,p}$) for VM_m running on server n_i during scheduling period p, with DRAM operating at refresh period $RP_{i,p}$ and DRAM temperature $TEMP$ as:

$$Metric = ML(Mem^R_m, Mem^{SLA}_m, RP_{i,p}, TEMP) \quad (12)$$

where ML is a trained ML model that predicts $Metric$, Mem^R_m is the average memory access rate of VM_m, and Mem^{SLA}_m is the size of memory allocated by VM_m.

The model uses a machine learning technique, the K-nearest neighbors algorithm, which was found to lead to a high DRAM error prediction accuracy [21]. To train the model, we collect $CE_{i,m,p}$ and $P^{UE}_{i,m,p}$ by executing different workloads under various refresh periods and temperatures. For each workload, we collect a set of inherent program features using a profiling tool as in [21]. However, rather than extracting numerous program features, we consider the memory access rate, which was found to have the highest impact on DRAM errors and the memory footprint of each application. Both are available from Google traces.

In our model, we predict $CE_{i,m,p}$ also taking into consideration the DIMM-to-DIMM variation, i.e the difference in DRAM error behavior across different chips. In particular, we use the obtained difference in $CE_{i,m,p}$ across DRAM devices to estimate the standard deviation of $CE_{i,m,p}$. To predict the number of CEs for a specific workload when DRAM operates under relaxed circuit parameters and temperature, we first predict the mean $CE_{i,m,p}$, and then build a Gaussian distribution using this mean and the obtained standard deviation. We randomly sample a $CE_{i,m,p}$ from the obtained Gaussian distribution, thus simulating the DIMM-to-DIMM variation. Figure 2 illustrates such a sampling procedure where the mean $CE_{i,m,p}$ (assuming the length $K \times SlotT$ of period p is 5 minutes) is predicted by the model for one of the executed benchmarks, i.e. the *memcached* workload, when DRAM operates under 2.283 s refresh period at $50°C$. We extend the model to automatically build the Gaussian distribution of $CE_{i,m,p}$ using the predicted mean and average standard deviation and randomly sample $CE_{i,m,p}$. For example, *sample 1* corresponds to $CE_{i,m,p} = 1800$ (the probability to sample this $CE_{i,m,p}$ is about 0.054), while *sample 2* corresponds to $CE_{i,m,p} = 2200$ (the probability to sample this $CE_{i,m,p}$ is about 0.053). We apply the same approach to consider the DIMM-to-DIMM variation when estimating $P^{UE}_{i,m,p}$.

B. Characterization Infrastructure

To collect the required datasets and train the model, we run compute intensive benchmarks (Rodinia, Parsec), data analytics benchmarks (Ligra) and Cloud workloads, such as memcached on a commodity 64-bit ARMv8-based server, the X-Gene2 Server-on-a-Chip. The server consists of eight 64-bit ARMv8 cores running at 2.4GHz and 4 Micron DDR3 8GB DIMMs operating at 1866 MHz. The server can support up to

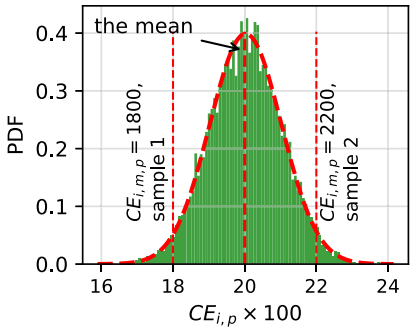

Fig. 2: Modeling $CE_{i,m,p}$ variation for different DRAM chips.

256GB of DRAM memory[1]. We measure the power of the SoC and DRAM using on-board sensors. The system implements ECC SECDED (single error correction, double error detection) and reports to the Linux kernel all detected CEs and UEs. We quantified the hardware and Linux kernel overhead to handle and report a single CE to be 100 μs on average. We use this DRAM error handling $overhead$ in Equation 2.

We perform the experiments under various RP values, ranging from 64ms to 2283ms. At relaxed operating points, we also operate DRAM at the minimum possible supply voltage of 1.428V (lower than the nominal 1.5V). Our study indicated that such a voltage reduces the power with a negligible impact on the number of DRAM errors under any temperature.

To train the DRAM error behavior model for different temperatures, we utilized a temperature-controlled testbed with heating elements, as discussed in our previous study [21]. We use three temperature levels: $50°C$, $60°C$ and $70°C$, that are common in dense server environments [7], [22], [23].

C. Modeling DRAM Power and Power at the Plug

1) DRAM power: DRAM power varies across different workloads. Figure 3a shows how the maximum (*Max*) and average DRAM power (*Average*) changes with the refresh period (under the lowest supply voltage). In this figure, *Max* depicts the maximum dynamic power observed in our experiments, which is incurred by the *pagerank* benchmark, while *Average* corresponds to the dynamic DRAM power averaged over all benchmark runs. We see that the maximum and average DRAM power can be reduced by 21 % and 30 %, respectively, when DIMMs operate under relaxed circuit parameters.

2) Power at the plug: Figure 3b compares the power savings that can be achieved for DRAMs and the entire system i.e. $Power(RP_{i,p}, u_{i,k})$ by scaling the refresh period under the lowered voltage. Power at the plug has been measured using a digital power meter. We see that the maximum savings are up to 3% if we measure power at the plug with a fully utilized CPU. In our experimental framework, we use only 4 DIMMs or 32 GB of memory in total. However, servers in modern data centers are equipped with many more DRAM devices

[1]In Section V we extrapolate to 512GB as well, in order to evaluate the effects on big-memory servers.

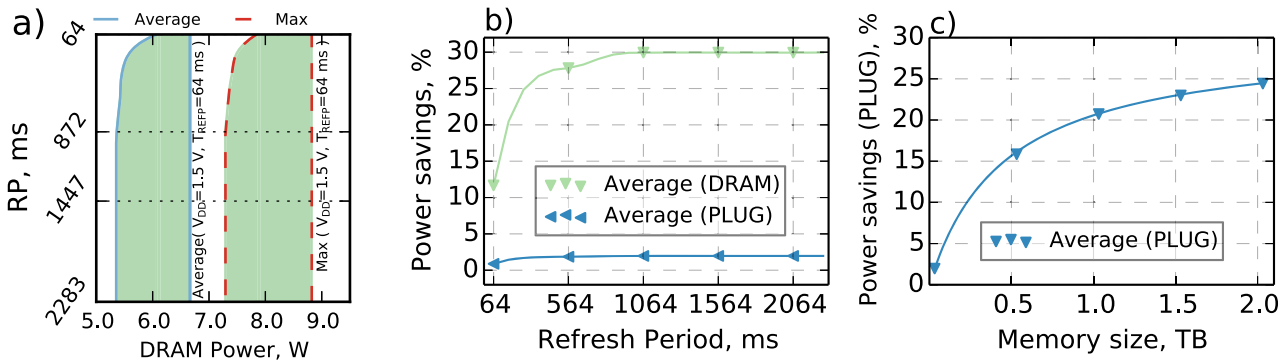

Fig. 3: a) The maximum and average DRAM power; b) Scaling of DRAM power savings with the refresh period; c) Scaling of DRAM power savings (at the plug) with memory size.

to process data. The provided power gain that we measure at the plug scales up with memory size and the number of DRAM devices used by servers. Figure 3c shows how the power savings scale with memory size, assuming full CPU utilization. To estimate the savings, we project the system power by increasing the number of DIMMs (up to 2TB). Note that the system power grows linearly with the number of DIMM devices [24]. We see that system power savings due to DRAM operation at reduced margins grow logarithmically with the size of memory (and the number of DIMM devices) and are in the order of 25% for a 2TB system.

The above examples are for the case where the CPU is fully loaded. However, in the general case, the system may be underloaded. The power model in the experimental evaluation also considers CPU utilization.

V. Experimental Evaluation

In this section, we present an experimental evaluation of *RM-DRAM* via simulations that quantify the trade-off between energy cost and potential SLA violation penalties. We resort to simulations as it is not feasible to obtain access to a large-scale installation with administrator privileges; the latter is necessary to set the DRAM refresh period beyond the nominal value.

A. Simulation Setup

For our simulations, we use the popular CloudSim toolkit [15]. CloudSim already supports several useful mechanisms such as VM consolidation, and it allows to configure many parameters of server nodes. We extended CloudSim to support configuration of the nodes at reduced margins by setting the DRAM refresh period and voltage, and to simulate CEs and UEs due to the operation of DRAM at reduced margins, using the memory model discussed in Section IV. We also implemented the *RM-DRAM* scheduling policy based on the cost model discussed in Section II.

As an indicative workload, we use a subset of Google traces [16] consisting of 10,000 VMs submitted for execution to the data center over 1 day. This corresponds to 288 scheduling periods / invocations of the scheduling policy (the scheduling period is 5 minutes). Each simulation is repeated 10 times and we report the average metrics across all 10 runs.

To evaluate the effect of DRAM operation at reduced margins, we consider the nominal refresh period of 64 ms and four indicative extended refresh period (RP) settings of 618, 1173, 1723 and 2283 ms, which can be chosen by our *RM-DRAM* policy to reduce the node's energy consumption. Moreover, we perform simulations with nodes of different DRAM capacity. Higher DRAM capacity may offer more energy gains, however the probability of error increases as nodes accommodate more VMs with a larger memory footprint. We explore four different data center scenarios, where nodes are homogeneous and all have 64, 128, 256 and 512 GB of DRAM, respectively.

Our policy receives from the traces the arrival/completion time and maximum CPU requirements CPU_m^{SLA} and memory footprint Mem_m^{SLA} for each VM_m and, for each scheduling period p, the VM's actual CPU requirements $CPU_{m,p}^{req}$ and memory access rate $Mem_{m,p}^{R}$ (Figure 1). The $CPU_{m,p}^{req}$, Mem_m^{SLA} and $Mem_{m,p}^{R}$ are forwarded to the DRAM error and system power model, together with the evaluated $RP_{i,p}$ setting for each node n_i considered to host VM_m. In turn, the model returns the expected number of CEs $CE_{i,m,p}$ and the probability of UE $P_{i,m,p}^{UE}$ for VM_m.

As discussed in Section IV, one more parameter that affects the number of errors is temperature, which is not directly controlled by the operator. Hence, it is important for a cloud policy to be adaptive in this respect. We consider the operating temperatures modeled in section IV: 50, 60 and 70°C.

We also use real values for the energy cost [25] and VM pricing [26]. The energy price in our experiments is \$0.15 per KWh and the VM price range for the types of VMs in the traces is \$0.0057 - \$0.4608 per hour, depending on the memory and core requirements of each VM.

We compare our *RM-DRAM* policy with the *LrMmt* policy [13], which performs VM consolidation. *LrMmt* uses local regression for identifying overloaded nodes and selects for migration the VMs with the minimum migration time. The adaptive heuristics employed by *LrMmt* for VM consolidation outperform the optimal online deterministic algorithm, there-

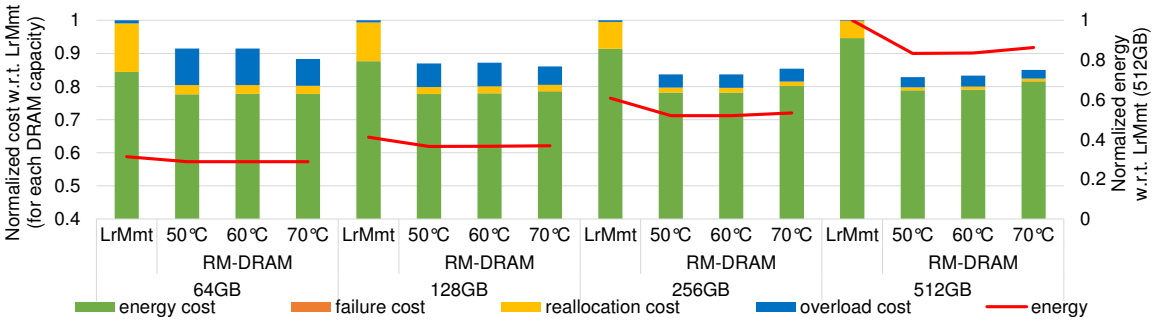

Fig. 4: Relative cost (left y-axis, bars, normalized w.r.t. *LrMmt* at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. *LrMmt* at 512GB), for effectively equal node utilization. Left y-axis starts at 0.4.

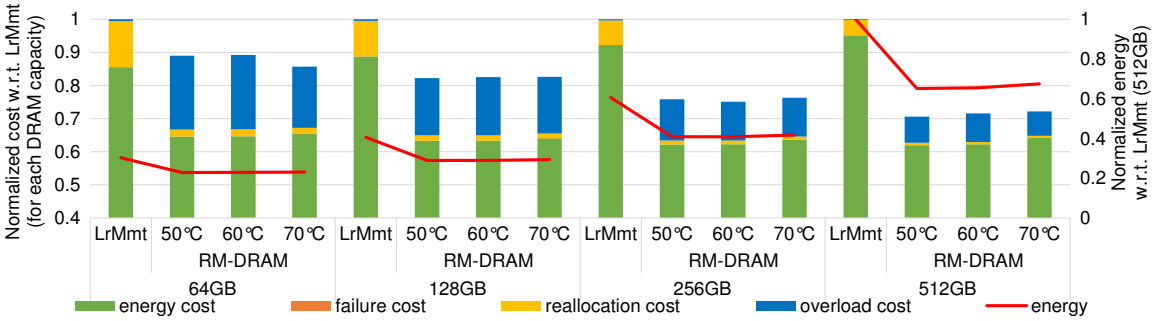

Fig. 5: Relative cost (left y-axis, bars, normalized w.r.t. *LrMmt* at the same memory capacity) and relative energy consumption (right y-axis, lines, normalized w.r.t. *LrMmt* at 512GB), for the same target node utilization. Left y-axis starts at 0.4.

fore *LrMmt* is an excellent challenger for *RM-DRAM*. *RM-DRAM* assumes stateless VMs and performs reallocation and restart. For fairness, we assume that the migration cost (for *LrMmt*) equals the restart cost (for *RM-DRAM*).

B. Results and Discussion

Both *RM-DRAM* and *LrMmt* take a target node utilization as a parameter. The goal of this parameter is to guard against node overloads. *LrMmt* blindly tries to respect the target utilization. The cost-aware VM consolidation policy of *RM-DRAM*, however, may violate the threshold – and risk paying SLA violation penalties due to overloads – if this is deemed to be a more cost-efficient option compared to reallocating and restarting excessive VMs and/or activating more nodes. Hence, *RM-DRAM* may end up with higher effective (actual) node utilization and less active nodes. This makes energy gains by DRAM operation at reduced margins hard to distinguish from energy gains due to more cost-effective VM consolidation.

In a first set of experiments, we focus on the gains obtained when DRAM is operated at reduced margins. We compare execution with approximately the same effective node utilization by *LrMmt* and *RM-DRAM* (which do not necessarily correspond to the same target utilization). More specifically, we manipulate the utilization target parameter to achieve an effective utilization of approximately 65% for each policy (a sweetspot between the optimal utilization targets of the two policies). This effectively factors-out the gains due to the more aggressive VM consolidation employed by *RM-DRAM*. Figure 4 illustrates the cost reduction and energy gains due to operation at reduced DRAM margins. The bars (left y-axis) correspond to the total cost for the data center normalized over *LrMmt* for each memory capacity. The cost is broken down to the cost of energy and the SLA violation penalty cost due to VM reallocations and restarts, node overloads and node failures. The red lines (right y-axis) correspond to the total energy cost of each configuration/policy normalized to the energy cost of *LrMmt* for nodes with 512GB DRAM. Note that one cannot compare the bars of different groups (memory capacities) in a direct way; to do so, one would have to scale their size relatively to the corresponding red line which shows the total cost w.r.t. a common reference (*LrMmt* for 512GB).

We observe that *RM-DRAM* successfully exploits the reduced margins of DRAMs. The energy gains w.r.t. the nominal DRAM configuration (represented by the green bars on *LrMmt*) depend more on nodes' DRAM capacity, rather than on temperature. They increase with higher DRAM capacity, reaching up to 16.69% for 512GB. In many cases, *RM-DRAM* approaches the highest feasible energy gains for the DRAM capacity at hand. Even in high temperatures, cost gains due to reduced margins are both feasible and significant.

In a second series of experiments, we study the combined effect of DRAM operation at reduced margins and VM consolidation, setting the same target utilization threshold (again 65%) for both applications. Figure 5 illustrates the cost benefit (bars) and energy benefit (lines) of *RM-DRAM* w.r.t. *LrMmt* for the four different memory capacity scenarios. We observe

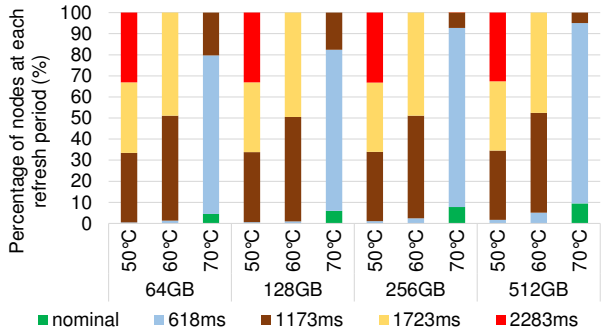

Fig. 6: Percentage of nodes configured to each refresh period setting (RP) for different DRAM capacities and temperatures.

a clear tradeoff between the cost of reallocating VMs to other nodes and the cost of node overloads. Contrary to *LrMmt*, *RM-DRAM* opts to increase the load of nodes – risking to overload them – thus decreasing both VM reallocations and the number of active nodes. *RM-DRAM* switches off, on average, 26.23% more nodes compared with *LrMmt* and achieves 34.84% energy reduction on average. 16.69% comes from reduced margins and the rest from cost-aware VM consolidation.

Our policy succeeds to reduce the cost for all DRAM capacities and temperatures w.r.t. *LrMmt*. The cost reduction ranges from 11.01% to 29.53% on average, depending on nodes' DRAM capacity and temperature. A larger DRAM capacity offers more opportunities for decreasing the cost thanks to configuration at reduced margins. A higher temperature increases the probability of errors, thus is associated with a higher probability of SLA violation penalty costs. *RM-DRAM* configures nodes more aggressively at reduced DRAM margins when the probability of errors is low, as in the case of 50 and $60°$C. However, it risks potential overloads due to error rate deviations related to application- and DIMM-to-DIMM variation. At $70°$C, where the expected error rate is higher and there is an increased probability for UEs, *RM-DRAM* is more conservative. As a result, it configures nodes so as to avoid crashes, however at the expense of slightly higher energy and total cost. The average percentage of node crashes in the simulations at $70°$C is just 0.0037%. No crashes were observed at lower temperatures.

Figure 6 shows the percentage of nodes configured to each of the supported RP settings for different DRAM capacities and temperatures. As expected, the most significant parameter affecting the selection of RP is temperature. At $50°C$ and $60°C$ almost all the nodes are configured at reduced margins, however at different refresh rates. At $50°C$, the three highest refresh periods are equally preferred. This is expected, as – at $50°C$ – they result to almost the same power gains, and very similar average error rates, making the error deviation due to application- and DIMM-to-DIMM-variance a dominant factor in the selection. At $60°C$ errors rates increase, and *RM-DRAM* avoids configuration at the highest RP, as it deems that energy gains will not outweigh the penalty costs. At $70°C$, where the probability of UEs also increases, *RM-DRAM* configures some of the nodes at the nominal settings, the other dominant refresh period being 618ms, which is the most conservative of the ones considered in the experimental evaluation.

VI. Related Work

Much of the previous research efforts on energy efficient cloud computing have proposed techniques such as power-aware VM consolidation to pack VMs to as few nodes as possible to enable unused nodes to go to a low-power mode [13]. More recent works introduce optimized VM consolidation algorithms that target the reduction of migrations [27], [28].

Modern processors support DVFS which is used to improve energy efficient task scheduling on servers that are either idle or operate under light workload conditions. [29] introduces a VM allocation algorithm for a DVFS-enabled cluster, whereas the authors in [30] exploit DVFS to reduce energy consumption for the execution of tasks with deadline constraints.

The previous techniques may adversely affect performance and Quality of Service (QoS) to the extent that they may break SLAs contracted to the Cloud provider. This is further complicated by the dynamically changing workloads and resource requirements. [14] combines DVFS with a VM consolidation policy to reduce power consumption while preventing performance degradation considering the DVFS configuration that would be necessary to maintain Quality of Service. In this work, to isolate the effects due to the exploitation of memory margins and cost-aware migrations (rather than due to CPU techniques for energy efficiency), we compare *RM-DRAM* against [13] (rather than [14]).

Going beyond DVFS, recent work has shown that the conservative CPU voltage margins can be exploited on top of VM consolidation to further improve energy efficiency, by selectively placing VMs on nodes where the CPU is undervolted in a controlled way to save energy without compromising performance [31]. In this work, we focus on DRAMs rather than CPUs. On CPUs, manifested errors usually result to node crashes or SDCs. Memory behavior is more graceful; the probability of errors can be modeled, and CEs manifest at rates that are orders of magnitude higher compared to UEs, for realistic operating temperatures and educated configuration at extended operating points. This, in turn, calls for a significantly different system model and the design of suitable workload- and system-management policies. Moreover, in this work, we combine DRAM operation at reduced margins with cost-aware, rather than threshold-based VM migration heuristics.

In our evaluation framework rather than using workload unaware DRAM error models as in existing studies [9], [11], [12], [32], we extend our recent workload aware DRAM error model with power values and retrain it to consider features that are highly correlated with DRAM reliability but can also be considered by the relevant input traces in CloudSim.

VII. Conclusions

In this paper, we introduce a VM scheduling policy that reduces the total energy cost and thereby increases the profit of data center providers by (i) operating the DRAM at reduced

margins in an educated way, and (ii) by applying cost-aware VM consolidation. The policy is based on a new architecture-agnostic system model that jointly considers memory and system level metrics. We evaluate the proposed approach using publicly available workload traces, power consumption values and estimated error rates for reduced margin operation coming from a memory reliability and power model that is trained using data from the characterization of a real, ARM-based server platform. Our results show that significant cost gains can be achieved compared with a state-of-the-art VM scheduling policy, up to 29.53% for large scale systems, even when operated at adverse temperature conditions. Those gains come from both reducing the excessive supply voltage and refresh period margins of DRAMs, and from applying cost- rather than threshold-based criteria for VM consolidation.

ACKNOWLEDGEMENT

This work has received funding from the European Commission, project UniServer, contract number 688540, project OpreComp, contract number 732631, and the European Union and Greek national funds through the Operational Program Competitiveness, Entrepreneurship and Innovation, under the call RESEARCH - CREATE - INNOVATE, project vipGPU, project code T1EDK-01149.

REFERENCES

[1] N. Engbers and E. Taen, "Green Data Net. Report to IT Room INFRA," *European Commission. FP7 ICT 2013.6.2;2014*, 2016.

[2] B. Giridhar, M. Cieslak, D. Duggal *et al.*, "Exploring DRAM organizations for energy-efficient and resilient exascale memories," in *SC '13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Nov 2013, pp. 1–12.

[3] I. Bhati, M. Chang, Z. Chishti *et al.*, "DRAM refresh mechanisms, penalties, and trade-offs," *IEEE Transactions on Computers*, vol. 65, no. 1, pp. 108–121, Jan 2016.

[4] M. E. Tolentino, J. Turner, and K. W. Cameron, "Memory MISER: Improving Main Memory Energy Efficiency in Servers," *IEEE Transactions on Computers*, vol. 58, no. 3, pp. 336–350, March 2009.

[5] T. Hamamoto, S. Sugiura, and S. Sawada, "On the Retention Time Distribution of Dynamic Random Access Memory (DRAM)," *IEEE Transactions on Electron Devices*, vol. 45, no. 6, pp. 1300–1309, Jun 1998.

[6] J. Liu, B. Jaiyen, Y. Kim *et al.*, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms," in *40th Annual International Symposium on Computer Architecture*, ser. ISCA '13, 2013, pp. 60–71.

[7] J. Liu, B. Jaiyen, R. Veras, and O. Mutlu, "RAIDR: Retention-Aware Intelligent DRAM Refresh," in *39th Annual International Symposium on Computer Architecture*, ser. ISCA '12, 2012, pp. 1–12.

[8] J. Stuecheli, D. Kaseridis, H. C.Hunter, and L. K. John, "Elastic Refresh: Techniques to Mitigate Refresh Penalties in High Density Memory," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '43, 2010, pp. 375–384.

[9] Y. Luo, S. Govindan, B. Sharma *et al.*, "Characterizing application memory error vulnerability to optimize datacenter cost via heterogeneous-reliability memory," in *IEEE DSN*, June 2014, pp. 467–478.

[10] X. Li and D. Yeung, "Application-Level Correctness and Its Impact on Fault Tolerance," in *2007 IEEE 13th International Symposium on High Performance Computer Architecture*, ser. HPCA '07, 2007, pp. 181–192.

[11] X. Li, M. C. Huang, K. Shen, and L. Chu, "A Realistic Evaluation of Memory Hardware Errors and Software System Susceptibility," in *2010 USENIX Conference on USENIX Annual Technical Conference*, ser. USENIXATC'10, 2010, pp. 6–6.

[12] A. Messer, P. Bernadat, G. Fu *et al.*, "Susceptibility of commodity systems and software to memory soft errors," *IEEE Transactions on Computers*, vol. 53, no. 12, pp. 1557–1568, Dec 2004.

[13] A. Beloglazov and R. Buyya, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 24, no. 13, pp. 1397–1420, 2012.

[14] P. Arroba, J. M. Moya, J. L. Ayala, and R. Buyya, "Dynamic Voltage and Frequency Scaling-aware Dynamic Consolidation of Virtual Machines for Energy Efficient Cloud Data Centers," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 10, p. e4067, 2017.

[15] R. N. Calheiros, R. Ranjan, A. Beloglazov *et al.*, "CloudSim: a Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," *Software: Practice and experience*, vol. 41, no. 1, pp. 23–50, 2011.

[16] C. Reiss, J. Wilkes, and J. L. Hellerstein, "Google Cluster-Usage traces: Format + Schema," Google Inc., Mountain View, CA, USA, Technical Report, Nov. 2011, revised 2014-11-17 for version 2.1. Posted at https://github.com/google/cluster-data.

[17] "AWS Graviton Processor. Enabling the best price performance in Amazon EC2," https://aws.amazon.com/ec2/graviton/.

[18] S. Garg, S. Gopalaiyengar, and R. Buyya, "SLA-based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter," *Algorithms and Architectures for Parallel Processing*, pp. 371–384, 2011.

[19] M. K. Qureshi, D.-H. Kim, S. Khan *et al.*, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," in *2015 45th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, June 2015, pp. 427–437.

[20] L. Mukhanov, K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis, "Characterization of hpc workloads on an armv8 based server under relaxed DRAM refresh and thermal stress," in *18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation*, ser. SAMOS '18, 2018, pp. 230–235.

[21] L. Mukhanov, K. Tovletoglou, H. Vandierendonck *et al.*, "Workload-Aware DRAM Error Prediction using Machine Learning," in *IEEE International Symposium on Workload Characterization (IISWC), Orlando, FL*, Nov 2019.

[22] L. Minas and B. Ellison, "The Problem of Power Consumption in Servers," http://www.drdobbs.com/the-problem-of-power-consumption-in-serv/215800830.

[23] L. Mukhanov, K. Tovletoglou, D. S. Nikolopoulos, and G. Karakonstantis, "DRAM Characterization under Relaxed Refresh Period Considering System Level Effects within a Commodity Server," in *2018 IEEE 24th International Symposium on On-Line Testing And Robust System Design (IOLTS)*, July 2018, pp. 236–239.

[24] Micron Technology, "MT18JSF1G72AZ-1G9 - 8GB," https://www.micron.com/products/dram-modules/udimm/part-catalog/mt18jsf1g72az-1g9, 2015.

[25] "Electricity price statistics," https://ec.europa.eu/eurostat/statistics-explained/index.php/Electricity_price_statistics.

[26] "Amazon EC2 pricing," https://aws.amazon.com/ec2/pricing/.

[27] C. Ghribi, M. Hadji, and D. Zeghlache, "Energy Efficient VM Scheduling for Cloud Data Centers: Exact Allocation and Migration Algorithms," in *13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing*, May 2013, pp. 671–678.

[28] Z. Cao and S. Dong, "An Energy-aware Heuristic Framework for Virtual Machine Consolidation in Cloud Computing," *The Journal of Supercomputing*, vol. 69, no. 1, pp. 429–451, 2014.

[29] G. von Laszewski, L. Wang, A. J. Younge, and X. He, "Power-aware Scheduling of Virtual Machines in DVFS-enabled Clusters," in *IEEE International Conference on Cluster Computing and Workshops*, Aug 2009, pp. 1–10.

[30] Q. Huang, S. Su, J. Li *et al.*, "Enhanced energy-efficient scheduling for parallel applications in cloud," in *12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, 2012, pp. 781–786.

[31] C. Kalogirou, P. K. Koutsovasilis, C. D. Antonopoulos *et al.*, "Exploiting CPU Voltage Margins to Increase the Profit of Cloud Infrastructure Providers," in *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, Larnaca, Cyprus*, May 2019, pp. 302–311.

[32] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM refresh-power through critical data partitioning," *SIGPLAN Not.*, vol. 46, no. 3, pp. 213–224, Mar. 2011.