2016 IEEE International Conference on Cloud Engineering

A Reinforcement Learning-Based Power Management Framework for Green Computing Data Centers

Xue Lin
Department of Electrical Engineering
University of Southern California
Los Angeles, CA
Email: xuelin@usc.edu

Yanzhi Wang
Department of Electrical Engineering
and Computer Science
Syracuse University
Syracuse, NY
Email: ywang393@syr.edu

Massoud Pedram
Department of Electrical Engineering
University of Southern California
Los Angeles, CA
Email: pedram@usc.edu

***Abstract*—Various power management techniques have been exploited to reduce the energy consumption of data centers. In this work, we propose a reinforcement learning-based power management framework for data centers, which does not rely on any given stationary assumptions of the job arrival and job service processes. By carefully designing the state space, the action space, and the reward of a learning process, the objective of the reinforcement learning agent coincides with our goal of reducing the server pool energy consumption with reasonable average job response time. Real Google cluster data traces are used to verify the effectiveness of the proposed reinforcement learning-based data center power management framework.**

***Keywords*-power management; reinforcement learning**

I. Introduction

In cloud computing, the capabilities of business applications are exposed as sophisticated services that can be accessed over a network. Cloud service providers are incentivized by the profits of charging the clients for accessing these services; clients are attracted by the opportunity for reducing or eliminating the costs associated with "in-house" provision of these services [1]. It is essential that the clients have guarantees from service providers on service delivery by Quality of Service (QoS) parameters and Service Level Agreements (SLAs).

Massive data centers have been built to accommodate the ever-growing demand for cloud services. However, data centers consume a large amount of electricity power. For instance, Microsoft data center in Quincy, Washington consumes 48 megawatts which is enough to power 40,000 homes [2]. According to [3], electricity used by data centers worldwide increased by about 56% from 2005 to 2010. Power management techniques have been employed for green computing to reduce data center power consumption and at the same time maintain QoS requirements.

In this work, we apply the reinforcement learning (RL) technique to the data center power management problem. Different from previous data center power management approaches, the reinforcement learning-based method does not rely on any given stationary assumptions of the job arrival and job service processes. Instead, by carefully designing the state space, the action space, and the reward, the objective of the RL agent coincides with the goal of reducing the server pool energy consumption while in the meantime achieving reasonable average job response time. The effectiveness of the proposed RL-based data center power management framework is verified by simulations using real Google cluster data traces.

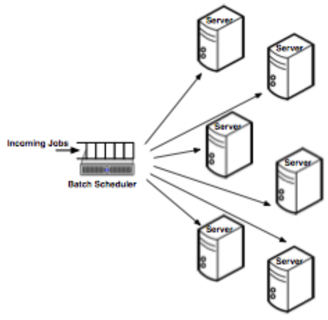

Fig. 1. The structure diagram of a server pool. The batch scheduler queues and dispatches the incoming jobs to the servers. The batch scheduler also performs the server-level resource consolidation.

II. System Modeling

We consider a data center consisting of several server pools. Jobs from data-center users are assigned to each server pool by the job broker of the data center. In this work, we focus on the power management problem of a single server pool involving job dispatch and resource consolidation.

Fig. 1 shows the structure diagram of a server pool consisting of a batch scheduler and a total number of N servers. The batch scheduler dispatches the incoming jobs to the servers in a first-come first-served manner. Whenever a new job arrives, the batch scheduler immediately assigns it to a server that has enough resource for processing the job. However, if there are no servers satisfying the resource requirements of this job at the moment, it will wait in the service queue of the batch scheduler until a server with enough resource is found.

Fig. 2 demonstrates how to dispatch jobs to a server.

978-1-5090-1961-8/16 $31.00 © 2016 IEEE
DOI 10.1109/IC2E.2016.33

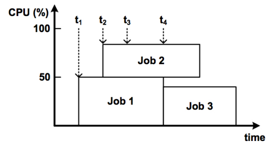

Fig. 2. An simplified example of job dispatch. Jobs 1, 2 and 3 arrive at time t_1, t_2, and t_3, respectively, with different CPU resource requirements. Jobs 1 and 2 are dispatched immediately after their arrivals. Job 3 is not dispatched until there is enough CPU resource available in the server.

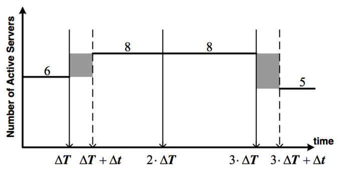

Fig. 3. The delay and power overhead in the resource consolidation. At time ΔT and $3\cdot\Delta T$, the batch scheduler makes decisions to turn on two servers and turn off three servers, respectively. It takes Δt time to finish the turning-on/off of the servers.

Suppose there are three jobs to be dispatched to a server. Jobs 1, 2, and 3 arrive at time t_1, t_2, and t_3 with CPU resource requirements of 50%, 30%, and 40%, respectively. Jobs 1 and 2 can be dispatched immediately after their arrivals, since there is enough CPU resource available for them at t_1 and t_2, respectively. However, Job 3 cannot be dispatched immediately after its arrival, and it waits until there is enough CPU resource available (i.e., t_4). The above simplified example shows how to dispatch jobs to one server. When it comes to the multiple-server scenario as in the server pool, the batch scheduler needs a low time complexity algorithm to select servers with enough resource available for dispatching the incoming jobs.

The batch scheduler also performs the server-level resource consolidation by setting active or sleep (e.g., the C3 mode in Intel processors) modes of the servers in the server pool. We employ a slotted time model for the resource consolidation decisions of the batch scheduler, i.e., at the beginning of each time slot with the length of ΔT, the batch scheduler makes the decision to turn on q more servers, where $q < 0$ means $|q|$ servers are turned off (from active to sleep). Please note that the job dispatch is performed at a finer time granularity than the resource consolidation, i.e., the batch scheduler dispatches jobs based on the job arrival time and the server resource availability, while the batch scheduler makes the resource consolidation decision on a fixed-length time slot basis.

We assume the power consumption of a server in the sleep mode is P_{slp}, and the power consumption of a server in the active mode is a function of the CPU utilization i.e., $P_{act}(CPU(t))$, where $CPU(t)$ denotes the CPU utilization of the server at time t. Please note that P_{slp} is much smaller than $P_{act}(0\%)$ [4]. We consider the delay and power overhead in the resource consolidation. As shown in Fig. 3, the batch scheduler decides to turn on two more servers at time ΔT. Then, it takes Δt for these two servers to be fully turned on, i.e., these two servers have a power consumption of $P_{act}(0\%)$ instead of P_{slp} during the time interval $[\Delta T, \Delta T + \Delta t]$, but these two servers cannot process jobs until time $\Delta T+\Delta t$. The length of Δt is in the order of seconds. At time $2\cdot\Delta T$, the number of active servers does not change, and therefore there are no delay and power overhead induced. At time $3\cdot\Delta T$, the batch scheduler decides to turn off three servers. Then, the jobs being processed by these three servers need to be migrated to the other active servers, and according to the Google cluster data trace [5], the migrated jobs will be moved to new servers by re-dispatching. These three servers cannot be fully turned off until time $3\cdot\Delta T + \Delta t$, i.e., these three servers have a power consumption of $P_{act}(0\%)$ during the time interval $[3\cdot\Delta T, 3\cdot\Delta T + \Delta t]$.

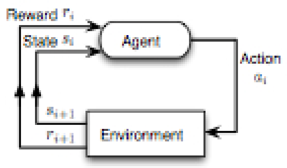

Fig. 4. The agent-environment interaction. At each decision epoch i, the agent observes the state s_i and takes an action a_i. Then at the next decision epoch $i+1$, it observes a new state s_{i+1} and receives a numerical reward r_{i+1}.

III. Server Pool Power Management Using Reinforcement Learning

The batch scheduler in a server pool controls the job dispatch and the server-level resource consolidation. The job dispatch is routinely performed with a finer time granularity based on the job arrival time and the server resource availability. Due to the large number of servers and the milli-second-level job inter-arrival time, the job dispatch algorithm can only be in low time complexity. Therefore, there is no much headroom left for server pool power management through the job dispatch. We solve the server pool power management problem through the server-level resource consolidation by using the reinforcement learning (RL) technique.

A. Reinforcement Learning Basics

Fig. 4 illustrates the agent-environment interaction in the reinforcement learning. Specifically, the agent and the environment interact at each of a sequence of decision epochs, i.e., $i = 0, 1, 2, 3, ...$ At each decision epoch i, the agent observes the environment's *state*, i.e., $s_i \in \mathcal{S}$, where $\mathcal{S}$ is the set of all possible states. According to some policy, the agent selects an *action*, i.e., $a_i \in \mathcal{A}$, where $\mathcal{A}$ is the set of all possible actions. Then, at the next decision epoch $i+1$, in part as a consequence of its action, the agent receives a numerical *reward*, i.e., r_{i+1}, and observes the environment in a new state, i.e., s_{i+1}.

A policy π of the agent is a mapping from each state $s \in \mathcal{S}$ to an action $a \in \mathcal{A}$ that specifies the action $a = \pi(s)$ the agent will select when the environment is in state s. The ultimate goal of the agent is to find the optimal policy, such that

$$V^{\pi}(s) = E\left\{\sum_{k=0}^{\infty} \gamma^k \cdot r_{i+k+1} \mid s_i = s\right\} \tag{1}$$

is maximized for each state $s \in \mathcal{S}$. The *value function* $V^{\pi}(s)$ is the expected return when the environment starts in state s at decision epoch i and follows policy π thereafter. γ is a parameter, $0 < \gamma < 1$, called the *discount rate* that ensures the infinite sum (i.e., $\sum_{k=0}^{\infty} \gamma^k \cdot r_{i+k+1}$) converges to a finite value. More importantly, γ reflects the uncertainty in the future. r_{i+k+1} is the reward received at decision epoch $i + k + 1$.

B. Reinforcement Learning Formulation of the Server-Level Resource Consolidation

In the server-level resource consolidation problem, the resource consolidation controller in the batch scheduler is the agent, and everything outside the controller in the server pool is the environment. The RL agent selects an action at each decision epoch, i.e., the controller makes a resource consolidation decision at the beginning of each time slot with the length of ΔT.

1) State Space of RL: To reduce the time complexity and accelerate the convergence rate of the RL algorithm, we propose a new definition of the state space with reduced cardinality by using fuzzy state representation as follows:

$$\mathcal{S} = \left\{s = [s_{job}, s_{act,svr}] | s_{job} \in \mathcal{S}_{job}, s_{act,svr} \in \mathcal{S}_{act,svr}\right\}, \tag{2}$$

where

$$\mathcal{S}_{job} = \{1, 2, ..., i, ..., N_{job}^{max}/K\}, \tag{3}$$

$$\mathcal{S}_{act,svr} = \{1, 2, ..., j, ..., N/M\}, \tag{4}$$

and N_{job}^{max} and N are the maximum number of jobs in the server pool and the maximum number of active servers, respectively. A state $s = [s_{job} = i, s_{act,svr} = j]$ describes that the number of jobs currently in the server pool is in the range $[(i-1) \cdot K, i \cdot K]$ and the number of active servers is in the range $[(j-1) \cdot M, j \cdot M]$, where K and M are factors of N_{job}^{max} and N, respectively. In this way, the cardinality of the state space is reduced to $O(\frac{N_{job}^{max}}{K} \cdot \frac{N}{M})$.

2) Action Space of RL: According to the state space definition, the action space of RL is defined by

$$\mathcal{A} = \left\{a = [q] | q \in \{-Q^{max}, ..., -1, 0, 1, ..., Q^{max}\}\right\}, \tag{5}$$

where an action $a = [q]$ taken by the RL agent is to turn on a number of q servers, and a maximum of $Q^{max} (< N)$ servers can be turned on/off simultaneously. Please note $q < 0$ corresponds to the case that $|q|$ servers are switched from active to sleep mode; $q > 0$ corresponds to the case that q servers are switched from sleep to active mode; and $q = 0$ corresponds to the case that the number of active servers does not change.

3) Reward of RL: The overall profit of the server pool is made up of the total revenue of processing all the incoming jobs minus the total energy cost. The income made by processing a job is inversely proportional to the response time of that job, including both waiting time in the service queue and processing time in the server. Therefore, we define the numerical reward r that the agent receives for taking action a while in state s as the negative of the weighted sum of the average job response time and the average power consumption of the server pool during that time slot i.e.,

$$r = -T_{job}^{avg} - w \cdot P_{pool}^{avg}, \tag{6}$$

where T_{job}^{avg} and P_{pool}^{avg} are the average job response time and the average power consumption of the server pool, respectively, and w is the relative weight of the power consumption to the response time.

The RL agent should be able to observe the reward r_{i+1} it receives for taking action a_i while in state s_i. In the reward definition by (6), the agent needs to measure T_{job}^{avg} and P_{pool}^{avg} values to decide the numerical reward. The P_{pool}^{avg} value, which is the average power consumption of the server pool during the time slot between two decision epochs (i.e., the current decision epoch i and the next decision epoch $i + 1$), can be measured directly from the server pool power meter. As for the T_{job}^{avg} value, which is the average response time of jobs being processed during the time slot between two decision epochs, cannot be obtained directly from measurement. According to Little's Law, the average number of jobs being processed is proportional to the average job response time. Therefore, we use the number of jobs the RL agent observes in the server pool at decision epoch $i+1$ to infer and approximate the T_{job}^{avg} value (similar approaches have been utilized in [6]).

C. Deriving the Optimal Power Management Policy

We have defined the state space, the action space, and the reward of the server-level resource consolidation problem. In this section, we will derive the optimal power management policy using the TD(λ)-learning algorithm [7], which has relatively higher convergence rate and higher performance in the non-Markovian environment. In the TD(λ)-learning algorithm, a Q value, denoted by $Q(s, a)$, is associated with each state-action pair (s, a), which approximates the expected discounted cumulative return of taking action a at state s. There are two basic steps in the TD(λ)-learning algorithm: *action selection* and *Q value update*. The pseudo code of the TD(λ)-learning algorithm for deriving the optimal power management policy is summarized in Algorithm 1, where lines 3-9 are the action selection step and lines 10-15 are the Q value update step.

IV. Experimental Results

In this section, we present experimental results of the proposed reinforcement learning-based power management approach on a server pool. We assume the following server pool setup for simulation: There are a total number of $N = 200$ servers in the server pool. We use the homogeneous

Algorithm 1 TD(λ)-Learning Algorithm

```
1:  Initialize $Q(s,a)$ arbitrarily for all the state-action pairs.
2:  for each decision epoch $i$ do
3:      Choose action $a_i$ for state $s_i$ using the exploration-exploitation policy:
4:      if in the exploration phase then
5:          select the current best action with probability $1-\epsilon$, select the other actions with equal probabilities.
6:      else
7:          select the current best action.
8:      end if
9:      Take action $a_i$, observe reward $r_{i+1}$ and next state $s_{i+1}$.
10:     $\delta \leftarrow r_{i+1} + \gamma \cdot \max_{a'} Q(s_{i+1}, a') - Q(s_i, a_i)$.
11:     $e(s_i, a_i) \leftarrow e(s_i, a_i) + 1$.
12:     for all state-action pair $(s,a)$ do
13:         $Q(s,a) \leftarrow Q(s,a) + \alpha \cdot e(s,a) \cdot \delta$.
14:         $e(s,a) \leftarrow \gamma \cdot \lambda \cdot e(s,a)$.
15:     end for
16: end for
```

server pool configuration: each server has the same amount of resources and the same power model (as a function of the CPU utilization). We assume the power consumption of a server in the sleep mode is $P_{slp} = 0$ W, and the power consumption of server in the active mode is $P_{act}(CPU(t)) = 150 + 150 \cdot CPU(t)$ W [8], where $CPU(t)$ is the CPU utilization of the server at time t. We set the server mode switching time $\Delta t = 5$ s, i.e., it takes 5 s for a server to switch from active (sleep) to sleep (active) mode, during which time the server cannot process any job but has a power consumption of $P_{act}(0\%)$ rather than P_{slp}. We assume the maximum number of servers that can be turned on/off simultaneously at one decision epoch $Q^{max} = 5$. The period of the resource consolidation decision epoch is $\Delta T = 60$ s.

We extract the job trace from the raw data tables of the Google cluster usage data trace [5], which provides Google cluster usage data over a month-long period in May 2011. The extracted job trace comprises information such as job arrival time (absolute time value), job execution time (relative time value), and the CPU request of each job (normalized by the CPU resource of one server). The extracted job trace orders jobs increasingly according to their arrival time. The average job arrival rate is about 216 jobs per minute.

We simulate the one-month operation of the server pool using the extracted job trace under our proposed reinforcement learning-based power management framework. Based on the convergence rate approximation of the TD(λ)-learning algorithm, the proposed power management policy can converge after three days' server pool operation. We present in Fig. 5 the total energy consumption for processing a one-day-long segment of the extracted job trace (left) and also the average job response time during this one-day segment of job trace (right). The 'Base' result is generated assuming all the servers are in active mode, whereas the 'Prop' results are generated by our proposed reinforcement learning-based

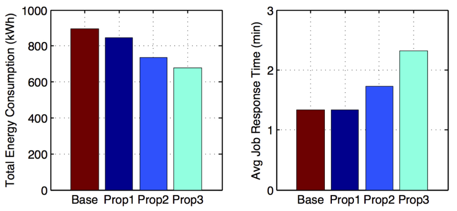

Fig. 5. The trade-off between energy consumption and job response time for a server pool with $N = 200$ servers. Different w parameter values are used to obtain the trade-off for the proposed RL-based power management framework (i.e., 'Prop1', 'Prop2', 'Prop3' results).

server pool power management framework using different w parameter values in the reward definition (6). By tuning the w parameter, the proposed power management framework can achieve a trade-off between the energy consumption and job response time. From Fig. 5, we observe that the proposed power management framework can achieve 5.4%, 17.8%, and 24.5% energy saving with 0%, 29%, and 73% job response time penalty, respectively.

V. Conclusion

We propose a reinforcement learning-based power management framework for data centers, which does not rely on any given stationary assumptions of the job arrival and job service processes. By carefully designing the state space, the action space, and the reward of a learning process, the objective of the reinforcement learning agent coincides with our goal of reducing the server pool energy consumption with reasonable average job response time.

Acknowledgment

This work is sponsored in part by a grant from the National Science Foundation.

References

[1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[2] R. H. Katz, "Tech titans building boom," *Spectrum, IEEE*, vol. 46, no. 2, pp. 40–54, 2009.

[3] J. Koomey, "Growth in data center electricity use 2005 to 2010," *A report by Analytical Press, completed at the request of The New York Times*, 2011.

[4] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 8, no. 3, pp. 299–316, 2000.

[5] J. Wilkes, "More Google cluster data." Google research blog, Nov. 2011. Posted at http://googleresearch.blogspot.com/2011/11/more-google-cluster-data.html.

[6] M. Ghamkhari and H. Mohsenian-Rad, "Optimal integration of renewable energy resources in data centers with behind-the-meter renewable generator," in *Proceedings of the IEEE International Conference in Commmunications (ICC)*, IEEE, 2012.

[7] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[8] E. Pakbaznia and M. Pedram, "Minimizing data center cooling and server power costs," in *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, pp. 145–150, ACM, 2009.