



WattValet: Heterogenous Energy Storage Management in Data Centers for Improved Power Capping

Shen Li, Shaohan Hu, Shiguang Wang, Siyu Gu, Chenji Pan, and Tarek Abdelzaher,
University of Illinois at Urbana–Champaign

https://www.usenix.org/conference/icac14/technical-sessions/presentation/li_shen

**This paper is included in the Proceedings of the
11th International Conference on Autonomic Computing (ICAC '14).
June 18–20, 2014 • Philadelphia, PA**

ISBN 978-1-931971-11-9

**Open access to the Proceedings of the
11th International Conference on
Autonomic Computing (ICAC '14)
is sponsored by USENIX.**

WattValet: Heterogeneous Energy Storage Management in Data Centers for Improved Power Capping

Shen Li, Shaohan Hu, Shiguang Wang, Siyu Gu, Chenji Pan, Tarek Abdelzaher

University of Illinois at Urbana-Champaign

{shenli3, shu17, swang83, siyugu2, cpan8, zaher}@illinois.edu

Abstract

This paper presents WattValet, an efficient solution to reduce data center peak power consumption by using heterogeneous energy storage. We henceforth call an energy storage device, a *battery*, with the understanding that the discussion applies to other devices as well such as pumped hydraulic and thermal systems. Previous work on energy storage management in data centers often ignores or underestimates their degree of heterogeneity. Even if batteries used in a data center are of the same model and purchased at the same time, differences in storing temperature and humidity, as well as discharging cycles and depth, gradually drive their characteristics apart. We show that differences in battery characteristics, such as discharge rates, if not fully accounted for, can lead to significantly suboptimal power caps. A new algorithm, called WattValet, is described that reduces peak power consumption by efficiently exploiting heterogeneity. Evaluation using Wikipedia traces shows that the power cap generated by WattValet is within 2% of the optimal solution, whereas WattValet finishes the computation orders of magnitude faster than the optimal solution even in small-scale experiments.

1 Introduction

With increased datacenter power consumption and growing concerns for sustainability of large-scale computing systems, reducing datacenter power becomes an increasingly important problem [9, 5, 17, 18, 4, 6]. Much recent work in both industry and academia addressed the challenge of shaving off power peaks using energy storage devices (*e.g.*, batteries) [7, 10, 16, 14]. Early work considered smarter charging and discharging strategies for energy storage systems [14, 16, 7, 2] to achieve improved power capping. However, they took into account only homogeneous environments, where all batteries are the same [14], or allowed for a very limited heterogeneity [7, 2]. In reality, even in homogeneous systems, battery characteristics gradually deviate from each other due to different storing temperatures, charging/discharging cycles, aging, and other factors. Hence, heterogeneity has to be considered explicitly.

Our work leverages the observation that load and power demand in data centers often follow clearly repeated patterns [3, 14, 15]. That is to say, by extracting

patterns from historical traces, future power demand can be predicted. In principle, given a power demand pattern and battery characteristics as input parameters, the optimal power capping problem can be formulated as a linear programming problem and solved using an LP solver. However, if the system contains hundreds of batteries, and thousands of time slots, LP solutions cannot guarantee to generate optimal results in a reasonable amount of time. For example, Wikipedia's trace shows a weekly pattern. If it is cut into 10-minute time slots, there will be more than 1000 time slots in the power demand pattern. Hence, clever approximations must be developed that significantly reduce computational overhead without tangibly degrading solution quality, which motivates this work.

The paper describes WattValet, an efficient solution to reduce datacenter peak power consumption using heterogeneous energy storage. WattValet takes a predicted power demand pattern and a set of heterogeneous batteries as input. It searches for a battery charging/discharging schedule that minimizes the power consumption peak. The search space is very large as it is a function of all variables indicating the amount of energy charged into/discharged from each battery at each time slot, and the starting time slot in the cyclic power demand pattern. The contribution of the paper lies in developing efficient heuristics that take into account not only battery capacity and efficiency but also maximum charge and discharge rates, leading to an improved power cap compared to prior art. Evaluation results show that the power cap achieved by WattValet is only 2% higher than the optimal value in the worst case, which significantly outperforms state-of-the-art solutions.

The remainder of this paper is organized as follows. The system model is described in Section 2. Section 3 elaborates the design of WattValet. Our solution is compared to the optimal solution as well as two others from recent literature in Section 4. Section 5 briefly summarizes related work. We conclude this paper in Section 6.

2 System Model

Datacenter workload often follows a clear periodic pattern. As an example, Figure 1 plots the English Wikipedia's workload in 2008 [15], presenting an obvious weekly pattern. Therefore, future workload can be predicted with accuracy from historical traces. Let P denote the power demand pattern. Hence, P can be represented by a repeating time series of period, T , composed

This work was sponsored in part by the National Science Foundation under grants CNS 13-20209, CNS 13-02563, CNS 10-35736 and CNS 09-58314.

of successive slots, such that the average demand in slot k is denoted by $P[k]$, where $k = 1, 2, \dots, T$.

Without an energy storage capability, the power cap P^c needs to be greater than the peak power consumption in the demand time series (i.e., $P^c \geq \max\{P[k] | k = 1, 2, \dots, T\}$). The presence of batteries can lower the power cap by charging during power demand valleys and discharging during power demand spikes.

Assume the datacenter is equipped with a centralized pool of B batteries [2] that serves the entire datacenter. We model each battery b with four parameters: maximum charging rate r_b^c , maximum discharging rate r_b^d , energy storage capacity c_b , and efficiency η_b . The maximum charging (discharging) rate represents the maximum amount of energy the battery can charge (discharge) in one time slot. The parameter c_b denotes the maximum amount of energy that battery b can store at any time. For each unit of energy spent on charging battery b , only η_b goes into the battery, which represents the battery efficiency. Please note, even if all batteries are of the same model, with the passage of time, their characteristics will gradually deviate due to differences in the environment (e.g., temperature, humidity, etc.) and usage (e.g., charging/discharging cycles and depth) [8].

We further assume that the batteries are equipped with on/off switches to connect or disconnect them for charging or for discharging purposes [2]. When a battery is connected it can charge or discharge at a rate no larger than its maximum charge or discharge rate. When a battery is disconnected it does not participate in charging or discharging.

The objective of this paper is to determine how much each battery should charge or discharge in each time slot such that the power cap is minimized, while being able to meet the power demand in every time slot. Let $u_b[k]$ represent the amount of energy that battery b is supplied (or, if negative, discharged) in time slot k . Hence, the solution sought in the paper is to compute $u_b[k]$ for all b and k , such that the demand is met and the power cap, P^c , is minimized. This can be modeled as a linear programming problem as shown below:

$$\begin{aligned}
 & \min_{P^c} \\
 & \text{s.t.} \quad \forall k : -r_b^d \leq u_b[k] \leq r_b^c \\
 & \quad \quad \forall k : x_b[k] + u_b[k] \geq 0 \\
 & \quad \quad \forall k : \sum_{b=1}^B u_b[k] - P^c + P[k] \leq 0 \\
 & \quad \quad x_b[0] = 0, \quad \forall k : x_b[k] \leq c_b \\
 & \quad \quad \forall k, P[k] > P^c : x_b[k] + \frac{u_b[k]}{\eta_b} = x_b[k+1] \\
 & \quad \quad \forall k, P[k] \leq P^c : x_b[k] + u_b[k] = x_b[k+1]
 \end{aligned} \tag{1}$$

where $x_b[k]$ is the amount of energy stored in battery b at time slot k . The first constraint guarantees that charging and discharging rates are not violated. The second constraint states that no battery can discharge more energy than it stores. The power balance equation at a

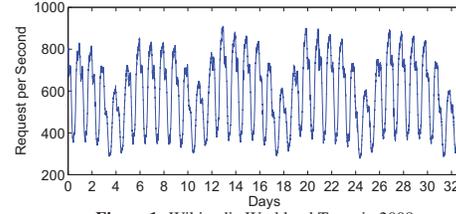


Figure 1: Wikipedia Workload Trace in 2008

given time slot is described with the third constraint. The fourth constraint enforces battery capacity. Batteries are initially empty ($x_b[0] = 0$). The last two constraints describe how the amount of energy stored in a battery is updated when the battery charges or discharges, respectively.¹

The above linear programming model contains a large number of variables and constraints. For example, if the data center is equipped with 1000 batteries and the weekly pattern is divided into 10-minute slots, this model will generate more than 2 million variables and more than 4 million constraints. LP solutions cannot guarantee to finish the computation in a reasonable amount of time. Hence, we seek an approximation with low worst-case asymptotic complexity.

3 System Design

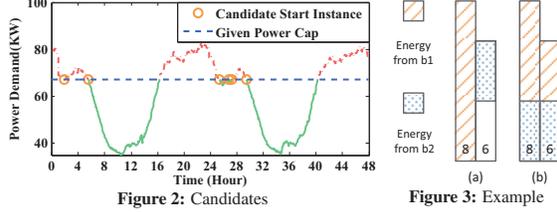
The original problem of minimizing the power cap can be transformed into a sequence of feasibility check problems. Each feasibility check needs to determine whether the predicted power demand time-series can be met for a given power cap P^c . A binary search can then find the minimum feasible power cap. Section 3.1 describes a feasibility check algorithm for a specified power demand sequence. It depends on which slot is considered to be the beginning of the sequence. Since the demand is cyclic, of period T , there are T candidate starting slots. Section 3.2 describes how to efficiently consider all possible start slots to find the best power cap overall.

3.1 Feasibility Check for a Given Power Demand Sequence

Let us call a time slot, a *charging* time slot if its average power demand is smaller than the power cap P^c . Otherwise, it is a *discharging* time slot. A set of consecutive charging (discharging) time slots is defined as a charging (discharging) *phase*. Hence, a power cap naturally divides a power demand sequence into alternating charging and discharging phases. We can refer to the j^{th} charging phase as set, CP_j , and to the i^{th} discharging phase as set, DP_i . As graphed in Figure 2, solid green curves represent the charging phases and dashed red curves represent the discharging phases. Figure 2 also suggests that

¹Note that, in principle, battery inefficiency affects both charging and discharging. Without loss of generality, we attribute all loss to charging, while modeling discharge as lossless. This does not change the result as long as capacity is reduced by discharge efficiency.

datacenter power demand does not change dramatically within small time intervals (*e.g.*, 5 minutes). Therefore, using the average power demand in a time slot, in lieu of the instantaneous demand curve is a good approximation.



Our algorithm alternates charging and discharging. Two main intuitions guide the design:

- *Intuition 1: Minimize lost energy:* Lost energy is energy that is wasted due to battery inefficiency. To minimize lost energy, charging should *exploit more efficient batteries first*. By exploiting them, we mean (i) charge them first and (ii) discharge them first, in order to incur minimum loss. Other less efficient batteries should be used only when the more efficient ones do not suffice.
- *Intuition 2: Minimize locked energy:* Locked energy is energy that is charged but not discharged within a given cycle. Specifically, the discharge rate of a battery limits the total amount of energy that it can discharge in different discharge phases. There is never a need to charge a battery beyond the maximum amount it can discharge, as such energy will in effect be “locked” and not utilized.

Note that, simple greedy algorithms that exploit the most efficient batteries first can result in some amount of locked energy, for example, if the maximum discharge rate of these batteries was low. In contrast, by putting a limit on how much each battery is allowed to charge (based on the amount it can ever discharge), we are able to outperform state of the greedy solutions. Below, we describe the algorithm in more detail.

3.1.1 Spatial Energy Allocation According to *Intuition 1*, we order batteries in decreasing order of efficiency, such that we consider the most efficient battery first. Let us number them $1, \dots, B$, such that $\eta_b \geq \eta_{b+1}$. For each battery, we need to determine how much to charge or discharge it in each time slot.

According to *Intuition 2*, we need to limit battery charging to the maximum amount that a battery can discharge. To compute the latter, it is first useful to define the notion of energy shortfall in slots, k , where demand $P[k]$ exceeds the power cap, P^c . The initial energy shortfall in a discharging slot k , denoted $\mathcal{P}^0[k]$, is given by the difference between the demand and the power cap. After

batteries $1, \dots, b \leq B$ have been allocated, the remaining shortfall becomes $\mathcal{P}^b[k]$. Hence:

$$\begin{aligned} \mathcal{P}^b[k] &= \mathcal{P}^{b-1}[k] + u_b[k] \\ \mathcal{P}^0[k] &= P[k] - P^c \end{aligned} \quad (2)$$

Let us now compute the maximum amount of energy a battery can discharge in a set of time slots, DP_i , belonging to the i th discharging phase. Let $md_b[i]$ be the maximum amount of energy that battery b may discharge in that phase, when the battery starts full. The amount can be computed as:

$$md[i] = \min \left\{ c_b, \sum_{k \in DP_i} \min(\mathcal{P}^b[k], r_b^d) \right\} \quad (3)$$

The equation states that the maximum energy discharged in each time slot by battery b is the minimum of the remaining shortfall in that time slot, $\mathcal{P}^b[k]$, and the maximum discharge rate, r_b^d . The maximum discharge over the entire phase is then the minimum of the discharge sum over all time slots, and battery capacity, c_b .

Next, we extend the result to multiple discharge phases. First, we define $mc_b[i][j]$ as the maximum amount of energy that battery b may carry from charging phase j (CP_j) to discharging phase i (DP_i), $j \leq i$, which is bounded by the summation of the amount of energy it may charge in each time slot and the remaining capacity. For $mc_b[i][i]$, the remaining capacity is the battery capacity c_b , as battery b has not been used in charging phase i before and there is no phases between charging and discharging phase i :

$$mc_b[i][i] = \min \left\{ c_b, \sum_{k \in CP_i} \min(-\mathcal{P}^b[k], r_b^c) \right\} \quad (4)$$

where we extend the definition of shortfall $\mathcal{P}^b[k]$ to the charging phases to denote the surplus energy. Hence, $me_b[i][i] = \min(md_b[i], mc_b[i][i])$ is the amount of energy battery b may carry from CP_i to DP_i . It becomes more complex if the charging phase j and discharging phase i are not consecutive (*i.e.*, $i > j$). Because, 1) battery b has been used to carry energy from $CP_{j'}$ to $DP_{i'}$, for all i' and j' such that $j \leq j' \leq i' \leq i$, occupying a portion of its capacity, 2) as battery b has been used in CP_j before when exploiting $DP_{i'}$, $j \leq i' < i$, we need to deduct $u_b[k]$ from its charging rate r_b^c in each time slot $k \in CP_j$, resulting in:

$$c'_b = c_b - \max \left\{ \max_{j \leq i' \leq i} \sum_{j'=1}^{i'} me_b[i'][j'] \right\} \quad (5)$$

$$mc_b[i][j] = \min \left\{ c'_b, \sum_{k \in CP_j} \min(-\mathcal{P}^d[k], r_b^c - u_b[k]) \right\} \quad (6)$$

where $\sum_{j'=1}^{i'} me_b[i'][j']$ in Equation (5) is the total energy battery b has discharged in $DP_{i'}$. Although this energy has been depleted in $DP_{j'}$, only $c_b - \sum_{j'=1}^{i'} me_b[i'][j']$ capacity can be used to carry energy

from charging phases before j' to discharging phases after j' . To calculate the remaining capacity for CP_j and DP_i , all discharging phases between j and i have to be accounted, leading to the remaining capacity c'_b .

The above analysis is the foundation of our algorithm, called the Discharge-bounded Highest Efficiency First (DHEF). The pseudo code is shown in Algorithm 1. The loop from Line 3 to 18 iterates over all discharging phases, where $|DP|$ denoting the total number of discharging phases. For each discharging phase, the algorithm iterates over all batteries according to the descending order of efficiency to carry energy from charging phases to the discharging phase. After calculating $me_b[i][j]$, it calls a function, EALLOC, to allocate the energy into each time slot in charging phase j . It computes $u_b[k]$ such that shortfall is updated correctly, using Equation (2). If the i^{th} discharging phase cannot be satisfied, line 15 exits the execution with return value null. Otherwise, it proceeds to meet all discharging requirement and return the battery scheduling plan in line 19.

Algorithm 1 Discharge-Bounded Highest Efficiency First

Require: Power demand sequence \mathcal{P} , battery set \mathcal{B}
Ensure: Battery scheduling plan \mathcal{U}

```

1: procedure DHEF( $\mathcal{P}, \mathcal{B}$ )
2:    $\mathcal{U} \leftarrow$  2D array of size  $|\mathcal{B}| \times |\mathcal{P}|$ 
3:   for  $i \leftarrow 1 \sim |DP|$  do
4:     for  $b \leftarrow 1 \sim |\mathcal{B}|$  do
5:        $\Sigma \leftarrow 0$ 
6:       calculate  $md_b[i]$ 
7:       for  $j \leftarrow i \sim 1$  do
8:         calculate  $me_b[i][j]$ 
9:          $me_b[i][j] \leftarrow \min(md_b[i] - \Sigma, me_b[i][j])$ 
10:         $\Sigma \leftarrow \Sigma + me_b[i][j]$ 
11:         $\mathcal{P}, \mathcal{U} \leftarrow$  EALLOC( $\mathcal{P}, CP_j, \mathcal{B}[b], me_b[i][j], \mathcal{U}$ )
12:      end for
13:       $\mathcal{P}, \mathcal{U} \leftarrow$  EALLOC( $\mathcal{P}, DP_i, \mathcal{B}[b], \Sigma, \mathcal{U}$ )
14:    end for
15:    if  $DP_i$  is not satisfied then
16:      return null
17:    end if
18:  end for
19:  return  $\mathcal{U}$ 
20: end procedure

```

Algorithm Analysis: The first 2 level loops of DHEF enumerate over all discharging phases and all batteries which contribute $\mathcal{O}(T)$ and $\mathcal{O}(B)$ computational complexity in the worst case. For each discharging phase, DHEF checks all charging phases prior to it, which induces at most $\mathcal{O}(T)$ computational complexity. Hence, the worst case computational complexity is $\mathcal{O}(T^2B)$.

It remains to show how to compute $u_b[k]$ such that shortfall is updated correctly, using Equation (2), as we consider each battery. This is described below.

3.1.2 EALLOC: Allocating One Battery The problem solved in function EALLOC is the following: Given an initial amount of available energy, $x_b[k]$ in battery, b ,

and given a discharge phase, compute the energy allocation $u_b[k]$ for battery b for each slot k in the discharge phase.

To appreciate why some allocations are better than others, consider Figure 3. Assume that two batteries, b_1 and b_2 , carry 8 and 6 units of energy respectively. Their discharge rates are 8 and 3. The batteries are discharged into two time slots, with a shortfall of 8 and 6 units of energy respectively. In Figure 3 (a), b_1 is used exclusively in time slot 1. Hence, it is depleted and cannot contribute to time slot 2. The discharge rate constraint of b_2 (namely, 3) falls short of supplying the needed power in slot 2 (namely, 6). Consequently, the schedule fails even though there is enough battery capacity to cover the shortfall. In contrast, Figure 3(b) is a solution where the shortfall is covered in all time slots. Specifically, in the first time slot, both batteries contribute (5 and 3 units of energy), leaving 3 units of energy in each battery. This is enough to cover the remaining shortfall in the second time slot.

The example demonstrates that one needs to be mindful of not only capacity but also the discharge constraints of batteries, as such constraints, if exceeded, will prevent covering the shortfall. Note that, these rate constraints apply independently in each time slot. Hence, given a set of undepleted batteries, the maximum shortfall slot determines the feasibility of meeting discharge rate constraints. If the shortfall in that slot is higher than the sum of the rate constraints, the allocation is infeasible.

The above observation suggests a simple solution to the problem of battery energy allocation; namely, allocate the energy across time slots such that the *maximum remaining shortfall is minimized*, hence maximally reducing the odds that rate constraints of remaining batteries will preclude filling the shortfall. The algorithm maintains a variable *bar* (the level to which to reduce the remaining shortfall). As *bar* is reduced, slots that reach the maximum discharge rate are “closed”. Shortfall in the remaining slots continues to be reduced to the same level (*bar*) until all capacity of battery, b , is exhausted. The resulting allocation value, $u_b[k]$, is then returned for each slot, k , as well as the updated shortfall, $\mathcal{P}^b[k]$.

3.2 Sequence Selection

Section 3.1 introduced algorithms to check feasibility under a given power demand sequence. Given that the power demand pattern is cyclic, one has to decide on a start time for the preceding algorithm to be applied. Naively checking all possible start slots multiplies the computational complexity by another $\mathcal{O}(T)$ term, resulting in an $\mathcal{O}(T^3B)$ overall computational complexity, which is undesirable. Below, we describe how to describe a more efficient way of considering all possible start times.

The curve in Figure 2 shows a power demand pattern generated using Wikipedia’s workload trace. The dashed horizontal line represents an attempting power cap P^c . As the power demand pattern is cyclic, fixing a starting instance is equivalent to selecting a sequence from the pattern. Let $\mathcal{D}_s^l = (P_s, P_{s+1}, \dots, P_{s+l-1})$ denote the power demand sequence starting from time instance s with length l . As batteries are all empty in the very beginning, feasible sequence does not start with discharging time slots. We can also exclude time slot k if $k-1$ is a charging time slot. Because, if P^c is feasible for \mathcal{D}_s^l , the same set of batteries is also able to satisfy \mathcal{D}_{s-1}^l under P^c . Therefore, remaining power sequence candidates start with intersection points of the power cap P^c and the power demand pattern. At last, we remove intersection points with positive derivatives on the power pattern curve, as no energy can be discharged from empty battery to meet the shortfall in discharging slot P_{s+1} . Hence, the feasibility checker only needs to try the intersection points whose derivatives are negative on the power demand curve, which are highlighted with circles in Figure 2.

Let \mathcal{C} denote the power demand sequence candidate set. Although $|\mathcal{C}|$ is much smaller than $|P|$, it is still not efficient enough. As shown in Figure 2, a two day trace generates 6 candidates with the given power cap. A one week trace may result in several tens of intersections. Before diving into refinements, we first discuss the composability of feasibility. Suppose there are two power demand sequences, $\mathcal{D}_{s_1}^{l_1}$ and $\mathcal{D}_{s_2}^{l_2}$. Define the sequence composition operation $|$ as:

$$\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2} = (P_{s_1}, \dots, P_{s_1+l_1-1}, P_{s_2}, \dots, P_{s_2+l_2-1}) \quad (7)$$

Please note that the operation $|$ is not commutative (i.e., $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2} \neq \mathcal{D}_{s_2}^{l_2} | \mathcal{D}_{s_1}^{l_1}$). Given feasibility check results of $\mathcal{D}_{s_1}^{l_1}$ and $\mathcal{D}_{s_2}^{l_2}$ under the same power cap P^c , can we tell whether P^c is feasible for $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$? If P^c is infeasible on $\mathcal{D}_{s_1}^{l_1}$, neither will it be feasible for $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$, as the first l_1 time slots in $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$ violate P^c anyway. If P^c is feasible for both $\mathcal{D}_{s_1}^{l_1}$ and $\mathcal{D}_{s_2}^{l_2}$, it will also be feasible for $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$. Because the first l_1 time slots in $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$ experience exactly the same situation as $\mathcal{D}_{s_1}^{l_1}$, and the last l_2 time slots in $\mathcal{D}_{s_1}^{l_1} | \mathcal{D}_{s_2}^{l_2}$ are no worse than $\mathcal{D}_{s_2}^{l_2}$ as it may or may not enjoy some leftover energy in batteries from the first l_1 time slots. For the last combination where P^c is feasible for $\mathcal{D}_{s_1}^{l_1}$ and infeasible for $\mathcal{D}_{s_2}^{l_2}$, we cannot tell its feasibility without invoking the feasibility checking algorithm (DHEF). The reason is that there might be some leftover energy after the first l_1 time slots which may help to meet shortfalls in the last l_2 time slots. Table 1 summaries the results, which we call the feasibility composition law.

The candidate set \mathcal{C} naturally divides the cyclic power demand pattern into an array of smaller pieces. Please note, the array is still cyclic, as we do not know the op-

$\mathcal{D}_{s_1}^{l_1}$	feasible	feasible	infeasible	infeasible
$\mathcal{D}_{s_2}^{l_2}$	feasible	infeasible	feasible	infeasible
$\mathcal{D}_{s_1}^{l_1} \mathcal{D}_{s_2}^{l_2}$	feasible	unknown	infeasible	infeasible

Table 1: Feasibility Composition Law

timal starting time slot yet. Let 1 denote feasible, and 0 denote infeasible. With a given power cap P^c , the feasibility checker tests each piece separately and generates a cyclic 0/1 series. According to feasibility composition law, compose two feasible or two infeasible pieces does not change the feasibility. Hence, we merge consecutive 0s into a single 0, and merge consecutive 1s into a single 1. Now, we have a new series with alternating 0 and 1. Again, based on the feasibility composition law, compose infeasible piece in front of feasible piece results in a larger infeasible piece, and the only unknown combination is (1, 0).

As the cost of evaluating a (1, 0) piece grows quadratically with its length, it is more efficient to avoid long (1, 0) pieces when possible. Therefore, we propose a greedy algorithm that only reduces the smallest candidate power demand piece in each iteration instead of reducing all of them. An example is shown in Figure 4.

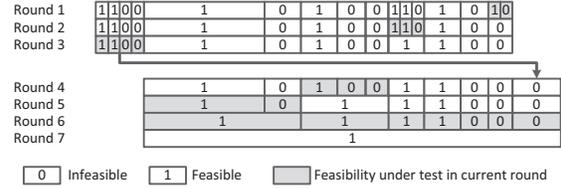


Figure 4: Accelerated Feasibility Check Example

Suppose we have n_k candidates in round k . Each round, the algorithm checks the (1, 0) subsequence with the smallest number of time slots. If the answer is feasible, this candidate merges with its right-hand side candidate, as all candidate subsequences start with a feasible piece. Otherwise, it merges with its left-hand side candidate, as all candidate subsequences end with an infeasible piece. Hence, we have $|\mathcal{C}| > n_k > n_{k+1} > 1$. Let W denote the length of the entire power demand pattern. Then, the length of the smallest candidate piece in round k is shorter than $\frac{W}{n_k}$. According to the computational complexity of the DHEF algorithm, it takes at most $\frac{1}{n_k} \mathcal{O}(T^2 B)$ to check the feasibility in k^{th} round. Therefore, the worst case computational complexity is:

$$\mathcal{O}(T^2 B) \sum_k \frac{1}{n_k^2} \leq \mathcal{O}(T^2 B) \sum_{i=1}^{|\mathcal{C}|} \frac{1}{i^2} \leq \frac{\pi^2 \mathcal{O}(T^2 B)}{6} = \mathcal{O}(T^2 B).$$

4 Evaluation

This section evaluates how WattValet compares to the optimal solution as well as state-of-the-art solutions in terms of approximating Optimality and handling Heterogeneity.

Battery configurations in our experiments are based on APC 3U UPS [1] devices. The designed UPS capacity is

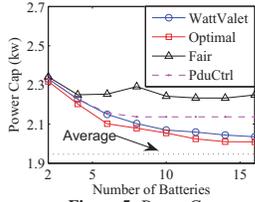


Figure 5: Power Cap

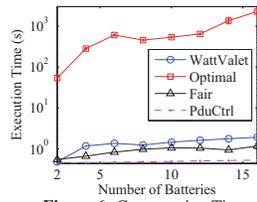


Figure 6: Computation Time

0.5 $KW \cdot H$, and its maximum charging and discharging rate are 240W and 2100W respectively. The efficiency is about 85% when serving more than 25% load. Aged batteries suffer degradations in their performance. We apply random degradations on batteries to generate a synthetic battery set. Each battery parameter is scaled with a random factor γ :

$$\gamma = random() \times \theta + 1 - \theta, \quad (8)$$

where $random()$ returns a random fractional number between 0 and 1 whenever called, and θ is an input parameter that controls the randomness bound. Wikipedia hosts its service on about 300 servers [13]. Wikipedia’s workload is translated into power consumption traces using the linear power model of our Dell D620 servers [11, 12].

WattVale trades optimality for efficiency. However, we need to make sure it stays within a reasonable range from the optimal solution. Evaluations compare WattVale to the optimal solution as well as two heuristics from recent literatures [2, 10]. The *fair* battery scheduling algorithm [2] deducts equal current from all batteries, whereas the *pduCtrl* algorithm [10] charges/discharges batteries one after another. Both solutions examine time slots chronologically. As the LP solver takes excessively long to converge, optimal solutions are only calculated for small scale evaluations. We scale down wikipedia’s workload to fit into 30 servers, and use at most 16 UPS devices. The other three-week trace in Figure 1 is used to generate the predicted power demand pattern by taking average in each time slot. The LP problem is solved with Matlab using *linprog* method. As plotted in Figure 5, the power cap achieved by WattVale is only 2% higher than the optimal solution in the worst case, while the other two heuristics lead to 7% and 12% degradations respectively. With the *fair* scheduler, the power cap increases when using 8 UPS devices. It is because the 8th battery suffers much lower efficiency compared to other batteries. The computation times of the four solutions are shown in Figure 6. The LP solver takes 1600 times longer than WattVale when using 16 batteries. The average power demand is plotted with the dotted line. There is still a gap between the settled optimal solution and average. The reason is that batteries’ efficiency is at most 85%. Some energy losses when carried from charging phases to discharging phases.

5 Related Work

Recently, shaving off data center power peaks using energy storage devices was introduced by Govindan *et al.* [7]. They gave a complete overview of the datacenter power hierarchy, and implemented a simple heuristic to reduce datacenter operational expenses. Later, Kontorinis *et al.* [10] adapted this idea to a Google’s datacenter, where each server is equipped with its own dedicated battery. Wang *et al.* [16] further investigated and evaluated broader types of energy storage methods, and compared their advantages and limitations. Other literature [14] explicitly achieved minimum power capping by modeling the problem as one of linear programming. However, all of above work considers only homogeneous batteries, or environments with very limited heterogeneity. In real-world systems, even if all batteries are identical initially, different storage temperature, humidity, and charging/discharging cycles will cause them to diverge over time.

This paper shares a similar infrastructure setup to Aksanli *et al.* [2]. Together with the utility power, a pool of batteries provide centralized support to the entire computing side. The major difference is that Aksanli [2] aims at maintaining homogeneity of all batteries which may not be possible in real world systems, whereas WattVale takes explicit advantage of battery heterogeneity.

Finally, the paper suggests that energy storage allocation in data centers is a QoS mechanism of growing importance at an age where sustainability of computation becomes a dimension of quality. The minimum power cap is presented as a metric of projected increasing interest. Solutions that lead to smaller power caps are more sustainable, because smaller power caps are indicative of smaller power consumption and better alignment between maximum and average power, achieved via more judicious energy storage management. The paper addresses the latter subproblem, where the cap is minimized for a given power demand profile.

6 Conclusion

This paper presents WattVale that reduces datacenter peak power consumption by using batteries. WattVale explicitly considers heterogeneities between batteries when generating the battery charging and discharging plan. It breaks down the power capping problem into two smaller parts, namely, generating battery charging/discharging plans for a given power demand sequence, and searching for the power demand sequence that leads to the minimum power cap. The efficiency of WattVale allows it to scale to datacenter-size problems, whereas the power capping result achieved by WattVale on Wikipedia’s data is within 2% of the optimal solution. WattVale considerably outperforms state-of-the-art solution, and the advantage increases as the heterogeneity grows.

References

- [1] Apc smart-ups on-line model SURTA3000XLTW. <http://www.apc.com>, Jan 2014.
- [2] B. Aksanli, E. Pettis, and T. Rosing. Architecting efficient peak power shaving using batteries in data centers. MASCOTS, 2013.
- [3] D. Beaver, S. Kumar, H. C. Li, J. Sobel, and P. Vajgel. Finding a needle in haystack: facebook’s photo storage. USENIX OSDI, 2010.
- [4] A. A. Bhattacharya, D. Culler, A. Kansal, S. Govindan, and S. Sankar. The need for speed and stability in data center power capping. IGCC, June 2012.
- [5] Gartner. Gartner says data center power, cooling and space issues are set to increase rapidly as a result of new high-density infrastructure deployments, May 2010. <http://www.gartner.com/it/page.jsp?id=1368614>.
- [6] D. Gmach, J. Rolia, C. Bash, Y. Chen, T. Christian, A. Shah, R. Sharma, and Z. Wang. Capacity planning and power management to exploit sustainable energy. CNSM, October 2010.
- [7] S. Govindan, A. Sivasubramaniam, and B. Urgaonkar. Benefits and limitations of tapping into stored energy for datacenters. ACM/IEEE ISCA, 2011.
- [8] S. Govindan, D. Wang, A. Sivasubramaniam, and B. Urgaonkar. Leveraging stored energy for handling power emergencies in aggressively provisioned datacenters. ACM ASPLOS, 2012.
- [9] J. Hamilton. Cost of power in large-scale data centers, November 2008. <http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx>.
- [10] V. Kontorinis, L. E. Zhang, B. Aksanli, J. Sampson, H. Homayoun, E. Pettis, D. M. Tullsen, and T. S. Rosing. Managing distributed ups energy for effective power capping in data centers. ISCA, 2012.
- [11] S. Li, T. Abdelzaher, and M. Yuan. TAPA: Temperature aware power allocation in data center with map-reduce. In *IEEE, IGCC*, 2011.
- [12] S. Li, H. Le, N. Pham, J. Heo, and T. Abdelzaher. Joint optimization of computing and cooling energy: Analytic model and a machine room case study. In *IEEE ICDCS*, 2012.
- [13] R. Miller. Google gift means more servers for wikipedia. <http://www.datacenterknowledge.com/>, Jan 2014.
- [14] D. S. Palasamudram, R. K. Sitaraman, B. Urgaonkar, and R. Urgaonkar. Using batteries to reduce the power costs of internet-scale distributed networks. ACM SoCC, 2012.
- [15] G. Urdaneta, G. Pierre, and M. van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. http://www.globule.org/publi/WWADH_comnet2009.html.
- [16] D. Wang, C. Ren, A. Sivasubramaniam, B. Urgaonkar, and H. Fathy. Energy storage in datacenters: what, where, and how much? ACM SIGMETRICS, 2012.
- [17] I. Widjaja, A. Walid, Y. Luo, Y. Xu, and H. J. Chao. Small versus large: Switch sizing in topology design of energy-efficient data centers. *IEEE IWQoS*, June 2013.
- [18] D. X, X. Liu, and B. Fan. Minimizing energy cost for internet-scale datacenters with dynamic traffic. *IEEE IWQoS*, June 2011.