The basic idea of the scheduler is a round robin scheduler where we introduce power constraints on tenants and the requests tenants can make. If a tenant's request is at the top of the queue and they've reached their power limit for the current power window, the request is put to the back of the queue. In order to prevent a device from monopolizing power in the system, if a request is at the top of the queue and the request has been made too many times in the current power window, the request is put at the back of the queue.

Each tentant t on the system is allocated a percentage c_t of power in a fixed power window P. For a basic implementation we can set $c_t = \frac{1}{N}$ where N is the number of tenants in the system. Note that we always have $\sum_{t=1}^{N} c_t = 1$. We can track the amount of estimated power used by each tenant in power windows P and dynamically update c_t without tracking much additional data as we already need each tenants estimated power usage to charge them.

For each operation tenants can request from a device, Sunneed intercepts the request and checks the estimated amount of power this operation will consume. This is checked by referencing the data from the regression model we will write to estimate power usage of each possible write to each device in the system.

Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of requests Sunneed can intercept. For each $x_i \in X$ let p_i denote the estimated power usage of the corresponding device operation obtained from regression model.

Let a_i denote the maximum number of times the request x_i can be made in the power window P. We need a_i such that $\sum_{i=1}^{n} a_i p_i \leq P$ so the system will not overdraw power in the power window P. The cap on the number of times each request can be made - a_i - ensures no one device or operation on a device is able to monopolize the power in the system.

We can start with $a_i = \frac{\frac{P}{f_i}}{p_i}$ where $f_i = |X|$, or the number of different requests Sunneed can intercept, for all i, giving each request equal percentage of the power in the power window P.

We can then keep track of the number of times each call is made in power windows of P. If the average number of calls made is lower than the current a_i for any x_i we can increase f_i so a_i = average number of calls, therefore lowering the percentage of power x_i gets in power window P. If any x_i is called a_i number of times frequently and we have $(a_i + 1)p_i \leq P - \sum_{j=1}^{n} p_j f_j : j \neq i$ – in other words, some other call x_j has had the corresponding f_j increased, making it possible to increase the percent of power allocated to x_i – we can set $f_i = 1 - \frac{P}{\sum_{j=1}^{n} p_j f_{j:j}}$.

This value of f_i gives us $a_i = \frac{P - \sum_{j=1}^{n} p_j f_{j:j}}{p_i}$, or the number of calls x_i that would put us at P power used in the power window P.

Problems with idea:

1. If a call x_i is made an average of 0 times in power window P we don't want to set $a_i = 0$ as this will prevent the request from being made in the future.

2. We'd have to track the number times each request is made to dynamically update the power allocation - this could potentially be a large amount of data to track and therefore take up memory. However, I don't think tracking tenant power usage to update c_t would present an issue as we already need to track tenants power usage and already have to maintain the number of power windows that have occurred to dynamically update a_i. Since the total power consumed by tenant and the number of power windows the tenant has existed for is all we need for an average, there is no additional data tracked.

Citations: Idea inspired by content of this paper (particularly section 2.2)

Utility Accrual Real-Time Scheduling Under the Unimodal Arbitrary Arrival Model with Energy Bounds(Haisang Wu; Binoy Ravindran; E. Douglas Jensen): https://ieeexplore-ieee-org.proxygw.wrlc.org/document/4302708