Online Multidimensional Load Balancing

Adam Meyerson *
Google, Inc.
awmeyerson@google.com

Alan Roytman
University of California, Los Angeles
alanr@cs.ucla.edu

Brian Tagiku †
Google, Inc.
btagiku@google.com

Abstract

Energy efficient algorithms are becoming critically important, as huge data centers and server farms have increasing impact on monetary and environmental costs. Motivated by such issues, we study online load balancing from an energy perspective. Our framework extends recent work by Khuller, Li, and Saha (SODA 2010) to the online model. We are given m machines, each with some energy activation cost c_i and d dimensions (i.e., components). There are n jobs which arrive online and must be assigned to machines. Each job induces a load on its assigned machine along each dimension. We must select machines to activate so that the total activation cost of the machines falls within a budget B and the largest load over all machines and dimensions (i.e., the makespan) by assigning jobs to active machines is at most Λ.

We first study the model in which machines are unrelated and can have arbitrary activation cost. In this problem, which we call Machine Activation, we extend previous work to handle jobs which arrive online. We consider a variant where the target makespan Λ and budget B are given. The first main result is an online algorithm which is $O(\log(md)\log(nm))$-competitive on the load Λ and $O(d\log^2(nm))$-competitive on the energy budget B. We also address cases where one parameter is given and we are asked to minimize the other, or where we want to minimize a convex combination of the two. Running our previous algorithm in phases gives results for these variants. We prove lower bounds indicating that the effect on the competitive ratio due to multiple phases is necessary.

Our second main result is in the same setting except all machines are identical and have no activation cost. We call this problem Vector Load Balancing, our objective is to minimize the largest load induced over all machines and dimensions (makespan) and the sum of the largest induced load on each machine (energy). We give an online algorithm that is $O(\log d)$-competitive on makespan, which improves even on the best prior offline result, and $O(\log d)$-competitive on the sum of induced loads if the target makespan is given; without this knowledge we show that it is impossible to get a competitive ratio independent of m.

*This work was done while the author was at the University of California, Los Angeles.
†This work was done while the author was at the University of California, Los Angeles.

1 Introduction

1.1 Motivation and Applications

Billions of dollars are spent every year to power computer systems, and any improvement in power efficiency could lead to significant savings [24]. With the rise of huge data centers and server farms, energy costs and cooling costs have become a significant expense as demand for computing power, servers, and storage grows. Indeed, energy costs and cooling costs are likely to exceed the cost of acquiring new hardware and servers. Managers of data centers wish to optimize power consumption without sacrificing any performance to minimize energy costs and cooling costs due to heat dissipation [21]. For these reasons, algorithms for energy efficient scheduling are very valuable, and even a small improvement could lead to significant savings and a positive impact on the environment.

As data centers and server farms grow in size, it becomes increasingly important to decide which machines should stay active and which should be shut down. In particular, there are opportunities for energy conservation and monetary savings due to cooling costs [21]. Moreover, these larger data centers have huge fluctuations in work loads, ranging from very high peaks to very low valleys. Hence, when demand is low, it is possible to shut down some machines which would allow for significant savings [16]. Certain jobs may need access to particular machines (due to data availability or device capability), and hence it makes sense to consider a subset of machines to activate for a given set of jobs. Once the appropriate machines have been activated, the pending jobs may then be scheduled. Note that the process of choosing which machines to activate is naturally an online problem, since jobs arrive dynamically over time.

In [23, 24, 26], methods were developed to measure power consumption at a high sampling rate. This allows us to measure energy requirements of recurring jobs at various speeds, and also to use machine learning techniques to estimate these requirements for new jobs. In [24] measurements were made for the energy effects of resource contention between jobs. The results indicated that jobs which make heavy use of different system components can be parallelized in a power-efficient manner, whereas jobs which make heavy use of the same components do not parallelize well. Their results motivate our belief that optimizing the scheduling of jobs to minimize power consumption is a non-trivial problem in real systems. From the standpoint of virtualization software such as VMware, it is important to effectively distribute jobs among machines so that resource contention (and thus energy use) is minimized. Multidimensional load balancing has applications in cutting stock, resource allocation, and implementation of databases for share-nothing environments [9, 10, 11]. This problem also has applications in multidimensional resource scheduling for parallel query optimization in databases. Query execution typically involves multidimensionality, particularly among time-sharing system resources such as the CPU or disk [10]. This motivates representing jobs as having d dimensions where each dimension represents the load induced by the job on the component. Real jobs often involve more than two components (and real network speeds depend on internet congestion). It is important to model the load placed on various system components (processor, network, memory, etc.) and the key to obtaining good performance (both in terms of completion times and power) is to balance the loads appropriately.

1.2 Problem Definitions

We consider two problems in this domain. Our first problem considers online allocation of jobs to unrelated machines with arbitrary activation costs and d dimensions, which we call the Machine Activation problem:

Definition 1 (Machine Activation Problem)**.** *We are given a set of m unrelated machines each with d dimensions (i.e., components such as CPU, memory, network) and an activation cost c_i. Moreover, a set of n jobs j arrive online, each inducing a load of p_{ij}^k if assigned to machine i on dimension k. We must select a set A of machines to activate such that $\sum_{i \in A} c_i \leq B$ for a constraint budget B, and assign jobs to active machines such that the total p_{ij}^k for jobs assigned to machine i along dimension k is at most a load constraint Λ.*

The main setting we study is when Λ and B are given to the algorithm. Our competitive guarantees hold if there is an offline integral solution with makespan Λ and budget B. Note that we also consider variants in which the load Λ or budget B may not necessarily be specified, in which case we seek to minimize the corresponding objective.

Our second problem considers online allocation of jobs to identical machines with multiple components and no activation costs.

Definition 2 (Vector Load Balancing Problem)**.** *We have m identical machines each with d components. Jobs $\vec{p}_j$ arrive online and are to be assigned to machines upon arrival. Here, the k'th coordinate p_j^k gives the load placed on component k by job j. Let ℓ_i^k denote the sum of p_j^k over all jobs j on machine i. The* load *ℓ_i of machine i is $\max_k \ell_i^k$. Our goal is to simultaneously minimize the makespan, $\max_i \ell_i$, and the energy, $\sum_i \ell_i$.*

1.3 Our Contributions and Techniques

For the Machine Activation problem, we design an online algorithm in Section 2 that is $O(\log(md)\log(nm))$-competitive on the load and $O(d\log^2(nm))$-competitive on the energy budget for the case when the load Λ and the budget B are given. It extends the result of [16] to an online setting and the work of [4] to the multidimensional setting. Our main technique considers the linear relaxation of an integer program. We approximately solve the linear relaxation online as constraints arrive, such that everything except the budget and load constraints are feasible. We develop a novel algorithm and technique for analyzing the primal linear program, with a unique combination of multiplicative and additive updates. This innovative approach ensures that key inequalities are feasible and keeps the total number of iterations of the algorithm small. Our fractional solution is $O(\log(md)\log(nm))$-competitive on the load and $O(d\log(nm))$-competitive on the energy budget. Our analysis includes a non-trivial potential function to obtain our competitive result on the load. We then describe an online randomized rounding scheme to produce a competitive ratio of $O(\log(md)\log(nm))$ on the load and $O(d\log^2(nm))$ on the budget. Combining our approach with the rounding scheme of the Generalized Assignment Problem [25] also gives an offline result similar to [16] with a substantially simpler rounding scheme. Since our problem generalizes both set cover even in the one-dimensional setting (by setting the processing times to 0 or ∞) and load balancing (by setting all c_i to 0), we can apply lower bounds from the online versions of these problems from [1, 2] to get polylogarithmic online lower bounds. We also show that no deterministic algorithm can be competitive.

In Section 3, we give upper and lower bounds for variants of the Machine Activation problem. We consider variants where one parameter (either B or Λ) is given up front and the goal is to minimize the other. We obtain our positive results by running our online algorithm from Section 2 in phases. Though this induces a logarithmic dependence on the value of the optimal solution, we show that such dependence is necessary for a fully online algorithm, suggesting a semi-online algorithm that is given a good estimate of the optimum performs much better. Lastly, we consider the case where neither parameter is given and the goal is to minimize a linear combination of the maximum load and the energy cost (say $c_\Lambda \Lambda^* + c_B B^*$, where Λ^*, B^* are the makespan and energy cost of a schedule, respectively). Our algorithm is $O(d\log^2(nm))$-competitive on this objective.

In Section 4, for the Vector Load Balancing problem, we design an online algorithm which is $O(\log d)$-competitive on the makespan, improving even the best known offline result [8]. In particular, the best previous offline result has not been improved upon for over a decade. Our algorithm is simultaneously $O(\log d)$-competitive on the energy usage if we are given a small piece of information (the maximum load induced by any single job on any single component). Without this information, we show our competitive ratios on the two criteria must have product $\Omega(\min(d, \log m))$. Our main technique involves adapting a result of Aspnes et al. [2] which works by assigning jobs greedily based on an exponential cost function. A direct application of their proof technique requires a near-optimum offline solution (which we do not have) and obtains competitive ratio $O(\log md)$, matching random assignment [8]. We modify their technique to make use of suitable rough bounds on the optimum load and exploit the fact that our machines are identical to obtain an $O(\log d)$ bound. Our analysis also includes the use of a non-trivial potential function argument.

1.4 Related Work

For the Machine Activation problem, the recent work of [16] studies the same problem in the offline setting. They give an algorithm for the unrelated Machine Activation problem which produces a schedule with makespan at most $(2+\varepsilon)\Lambda$ and activation cost at most $2(1+\frac{1}{\varepsilon})(\ln \frac{n}{OPT}+1)B$ for any $\varepsilon > 0$ (where OPT is the number of active machines in the optimal solution), assuming there is a schedule with makespan Λ and activation cost B. They also give a polynomial time approximation scheme for the uniformly related parallel machines case (where machine i has speed s_i and $p_{ij} = \frac{p_j}{s_i}$), which outputs a schedule with activation cost at most B and makespan at most $(1+\varepsilon)\Lambda$, for any $\varepsilon > 0$. In [19], a generalized version of the Machine Activation problem is considered in the offline setting where each machine's activation cost is a function of the load assigned to the machine. An algorithm is given which assigns at least $n-\epsilon$ jobs fractionally with cost at most $(1+\ln(n/\epsilon))OPT$.

In addition, [19] studies an offline version of the Machine Activation problem in which each machine has d linear constraints. For this version, they give a solution which is $O(\frac{1}{\epsilon}\log n)$ times the optimal activation cost while breaking the d machine constraints by at most a factor of $2d+\epsilon$. This can be compared with our online, multidimensional guarantees of $O(\log(nm)\log(md))$ on the load and $O(d\log^2(nm))$ on the activation cost. There is also the recent work of [4], which we extend to the multidimensional setting. The work of [4] studies several problems. For their generalized framework, which is referred to as the Online Mixed Packing and Covering (OMPC) problem, a deterministic $O(\log P \log(v\rho\kappa))$-competitive algorithm is given, where P is the number of packing constraints, v is the maximum number of variables in any constraint, and ρ (respectively, κ) is the ratio of the maximum to the minimum non-zero packing (respectively, covering) coefficient respectively. Hence, if all coeffients are either 0 or 1, this is $O(\log P \log v)$-competitive. Note that we cannot simply apply the general scheme in [4] for OMPC, since our packing constraints are not given offline (indeed, this is precisely where the online nature of our problem comes into play). The OMPC framework only models programs in which packing constraints are given offline. The work of [4] also studies a problem called Unrelated Machine Scheduling with Startup Costs (UMSC), which is similar to our problem in the single dimensional case. In particular, when $d = 1$, they give an $O(\log m)$-competitive result on the makespan and an $O(\log(mn)\log m)$-competitive result on the energy budget. We extend this result to the multidimensional setting. Note that it is not clear how to adapt their algorithm to the multidimensional setting, and we develop our own framework which uses a novel combination of additive and multiplicative updates in our fractional algorithm.

In [2], they consider the online load balancing problem without activation costs. They give an $O(\log m)$-competitive algorithm for unrelated machines and an 8-competitive algorithm for related machines. In [3], they consider the online load balancing problem without activation costs where

the load on a machine is measured according to the L_p norm. Their main result is that the greedy algorithm is $O(p)$-competitive under the L_p norm, and any deterministic algorithm must be $\Omega(p)$-competitive. In [25], they give the first constant approximation (offline) for the unrelated machines case. In [7], the identical machines case is studied in the online setting without activation costs. It is shown that greedy is globally $O(\log m)$-balanced in the restricted assignment model and globally $O(\log m)$-fair in the 1-∞ model (see [7] for details). Our problem and techniques are substantially different, as we do not design or analyze a greedy algorithm. For a survey on power management and energy minimization, see [14]. Also, see [22] for a comprehensive survey on online scheduling.

For the Vector Load Balancing problem, the single-dimensional case has an offline PTAS [13] and a $(2-\varepsilon)$-competitive online algorithm for a small fixed ε [6]. The multidimensional version was introduced by [8], which includes an offline PTAS with running time exponential in the number of dimensions d and an offline $O(\log^2 d)$-approximation with polynomial running time. They also prove that a simple randomized algorithm is $O(\frac{\log md}{\log\log md})$-competitive (with m machines and d dimensions), and that no polynomial-time offline algorithm can attain a constant-factor approximation under standard complexity assumptions.

Most of the prior work (including a substantial portion of [8]) focuses on vector bin packing where we have a *hard constraint* on the makespan and must minimize the number of machines (bins). Some of these results include [5, 8, 12, 15, 17, 18, 20] with the best being $O(\log d)$-approximations.

2 Machine Activation

2.1 LP and Algorithm

We formulate the problem as an integer program, where $y_i = 1$ means machine i is activated, and $x_{ij} = 1$ means job j is assigned to machine i. We assume the target load Λ and budget B are given, and that there is an offline integral solution with makespan at most Λ and budget at most B.

1. For all $1 \le i \le m$, we have $0 \le y_i \le 1$.
2. For all $1 \le i \le m$ and $1 \le j \le n$ we have $x_{ij} \ge 0$.
3. For all j, we have $\sum_{i=1}^{m} x_{ij} \ge 1$.
4. For all i, j we have $x_{ij} \le y_i$.
5. For all i, k, we have $\sum_{j=1}^{n} x_{ij} p_{ij}^k \le \Lambda y_i$.
6. We have $\sum_{i=1}^{m} y_i c_i \le B$.

Our goal is to design an online algorithm to solve the integer version of this linear system, while violating constraints 5 and 6 by at most a bounded factor. We will do this by first providing an online solution to the linear relaxation above, which may also violate the first constraint by possibly having $y_i \ge 1$, then describe an online rounding technique to produce an integer solution in Section 2.3. We will assume that either $\frac{B}{m} \le c_i \le B$ or $c_i = 0$ for all i (if more then discard machine i as offline cannot use it, if less then simply buy machine i and assume $c_i = 0$ for a constant factor increase in the total cost). Note that we can normalize B to anything we wish - we will choose $B = \Theta(m)$ so that any nonzero c_i is at least a constant (for instance, at least 1). We define $q_{ij}^k = p_{ij}^k/\Lambda$ and $\ell_i^k = \sum_j q_{ij}^k \max\{x_{ij} - \frac{1}{jm}, 0\}$. Let $a \ge 1$ be a constant to be set later.

We first give some intuition for our algorithm. When a job j arrives, most constraints are satisfied except for $\sum_{i=1}^{m} x_{ij} \ge 1$. To fix this, we need to raise the x_{ij} variables until this inequality is satisfied. However, this may cause other inequalities such as $x_{ij} \le y_i$ and $\sum_{j=1}^{n} x_{ij} p_{ij}^k \le \Lambda y_i$ to be violated. Hence, we will only increase x_{ij} if $x_{ij} \le y_i$ will continue to hold (if we do raise x_{ij}, then we also increase y_i to satisfy the load inequalities). If increasing x_{ij} would cause $x_{ij} > y_i$, then we simply increase y_i. We increase x_{ij} multiplicatively, so that the larger x_{ij} is, the larger the increase.

Moreover, it seems intuitively clear that we should increase x_{ij} for job j less aggressively if p_{ij}^k, c_i, or ℓ_i^k is large (in fact, we penalize machine i with an exponential cost function for the load ℓ_i^k in order to obtain our competitive ratio on the load constraints). We also increase y_i multiplicatively whenever it is too small (again, intuitively, y_i should be increased less aggressively if c_i is large). When a variable is small, it may take many iterations for multiplicative updates to increase its value substantially. We use additive updates to avoid this issue, and couple the additive updates with multiplicative ones to achieve our feasibility and competitive guarantees by keeping the number of iterations in our algorithm small. See Algorithm 1 for details.

```
1 Initialize x_ij ← 0 for all i, j and y_i ← 0 for all i
2 When job j arrives, set x_ij ← 1/(jm) for each i such that p_ij^k ≤ Λ for all k and y_i ← y_i + 1/(jm) for
  all i
3 while job j has Σ_{i=1}^m x_ij < 1 do
4     for each 1 ≤ i ≤ m : p_ij^k ≤ Λ for all k do
5         Set z_ij ← x_ij / (Σ_k p_ij^k (a^{ℓ_i^k} + c_i) + 1)
6         if x_ij + z_ij ≤ y_i then
7             Set x_ij ← min{x_ij + z_ij, 1} and y_i ← y_i + z_ij max_k q_ij^k
8         else
9             Set y_i ← y_i(1 + 1/(dΛc_i))
```

Algorithm 1: Fractional Assignment.

2.2 Analysis

We prove the following theorem regarding feasibility properties in Appendix A.1.

Theorem 1. *Inequalities* 3, 4, *and* 5 *of the linear program are satisfied.*

Each time through the loop at line three of the algorithm will be called a **reinforcement** step. Let r_j represent the number of reinforcement steps which occur on the arrival of j.

Lemma 1. *For $a \geq 1$, when j arrives, the r_j reinforcement steps increase $\sum_{i=1}^m \sum_{k=1}^d a^{\ell_i^k}$ by at most $\frac{a-1}{\Lambda} r_j$.*

Proof. Each reinforcement step increases ℓ_i^k by at most $z_{ij} q_{ij}^k$ for each i and k. Thus the total increase in the summation is bounded by $\sum_{i=1}^m \sum_{i=1}^k a^{\ell_i^k}(a^{z_{ij} q_{ij}^k} - 1)$. By the definition of z_{ij}, we will always have $z_{ij} q_{ij}^k \leq 1$. In general for any $a \geq 1$ we will have $a^x - 1 \leq (a-1)x$ whenever $0 \leq x \leq 1$, and applying this allows us to bound the summation by $\sum_{i=1}^m \sum_{k=1}^d a^{\ell_i^k}(a-1)(z_{ij} q_{ij}^k)$. We can substitute $z_{ij} \leq \frac{x_{ij}}{\sum_k p_{ij}^k a^{\ell_i^k} + 1}$ and use the fact that $\sum_i x_{ij} < 1$ prior to the reinforcement to get that $\sum_{i=1}^m \frac{a-1}{\Lambda} x_{ij} \leq \frac{a-1}{\Lambda}$. □

Lemma 2. *The total number of reinforcement steps is $\sum_j r_j \leq \Lambda(\log nm)(\sum_i \sum_k a^{\ell_i^k} + 2dB) + n(1 + \log mn)$.*

Proof. We split the reinforcements occurring on the arrival of j into two sets. Suppose i is the machine to which j is assigned by the optimum solution. We have $\bar{r}_j$ **load reinforcing** steps where $x_{ij} + z_{ij} \leq y_i$ and $\hat{r}_j$ **cost reinforcing** steps where the opposite was true. Clearly $r_j = \bar{r}_j + \hat{r}_j$.

Each load reinforcing step increases x_{ij} by a factor of at least $1 + \frac{1}{\sum_k p_{ij}^k (a^{\ell_i^k} + c_i) + 1}$ (except for possibly the last and only reinforcement step for job j in which x_{ij} is set to 1 instead of $x_{ij} + z_{ij}$ at line seven). Thus every $\sum_k p_{ij}^k (a^{\ell_i^k} + c_i) + 1$ such steps increase x_{ij} by a constant factor. Since x_{ij} is initially at least $\frac{1}{mn}$, the total number of such steps is bounded by $\log(nm)(\sum_k p_{ij}^k (a^{\ell_i^k} + c_i) + 1) + 1$. Summing over all machines i used by the optimum solution, we get $\sum_j \bar{r}_j \leq \Lambda \log(nm)(\sum_{i \in OPT} \sum_k a^{\ell_i^k} + dB) + n \log(mn) + n$.

Consider all cost reinforcing steps for jobs which the optimum assigns to machine i. The initial value of y_i is $\frac{1}{m}$. Each time we apply a cost reinforcing step, we increase this by a multiplicative $1 + \frac{1}{d\Lambda c_i}$. Note that we will never have $x_{ij} + z_{ij} \geq 2$, as we would not perform a reinforcing step unless $x_{ij} < 1$, from which $z_{ij} < 1$ follows. Thus, to perform a cost reinforcing step we must have $y_i \leq 2$. It follows that the total number of cost reinforcing steps performed for j with optimum assignment i is at most $d\Lambda c_i \log 2m$. If we sum this over all active machines i in the optimum solution (and observe that the optimum solution must not exceed the budget B) we get $\sum_j \hat{r}_j \leq dB\Lambda \log 2m$. Combining the equations along with the assumption $n \geq 2$ gives the lemma. □

Lemma 3. *If we set $a = 1 + \frac{1}{2 \log nm}$, then the final value of the potential function $\Phi = \sum_i \sum_k a^{\ell_i^k}$ after all jobs have arrived is at most $3md + 2dB$.*

Proof. Initially, the potential function equals md (since all loads $\ell_i^k = 0$). The function Φ increases with each reinforcement step, so by Lemmas 1 and 2, the final value is $\Phi \leq md + \frac{a-1}{\Lambda} \sum_j r_j \leq md + \frac{a-1}{\Lambda}(\Lambda \log(nm)(\sum_i \sum_k a^{\ell_i^k} + 2dB) + n(1 + \log mn))$. For $a = 1 + \frac{1}{2 \log nm}$, this implies that the final value is $\Phi \leq 3md + 2dB$ (we can normalize Λ to anything, so we choose $\Lambda = \Theta(n)$). □

Theorem 2. *For $a = 1 + \frac{1}{2 \log nm}$, we have $y_i \leq [\frac{\log n}{m} + 4 + 2 \log(nm) \log(3md + 2dB)]$ for all i.*

Proof. Applying Lemma 3 along with the value of a specified in the theorem, we know for any k that $\sum_i a^{\ell_i^k} \leq 3md + 2dB$ (where ℓ_i^k is the final load on machine i). Since each term in the summation is non-negative, we can bound the ℓ_i^k values by taking the log of both sides as $\ell_i^k \leq 2 \log(nm) \log(3md + 2dB)$.

We observe that y_i increases at several points in the algorithm. The total increase at step two of the algorithm can be at most $\frac{\log n}{m}$. The increases at line nine can only occur if $y_i \leq 2$, and will not cause y_i to exceed four (since we can ensure $1 + \frac{1}{d\Lambda c_i} \leq 2$ by scaling Λ and B appropriately). The increases at line seven always increase ℓ_i^k by the same amount as y_i, so the total increase in y_i due to these steps is at most ℓ_i^k, which is bounded as above. Combining these gives the result. □

Theorem 3. *The algorithm maintains $\sum_i y_i c_i \leq B \log n + (6md + 8dB + 4)(\log nm)$.*

Proof. Initially the left side of the equation is zero. When j arrives, every y_i increases by $\frac{1}{jm}$. Since each y_i has $c_i \leq B$ (otherwise we drop that machine), the total increase in cost due to this is at most $\frac{B}{j}$. Thus in total all arrivals increase the cost by at most $\sum_j \frac{B}{j} = B \log n$.

We also increase the y_i values when we perform a reinforcing step. Some y_i values increase by an additive $z_{ij} \max_k q_{ij}^k \leq \frac{x_{ij}}{\Lambda c_i}$, while others increase by a multiplicative $1 + \frac{1}{d\Lambda c_i}$. The increase in cost due to additive increases is at most $\frac{\sum_i x_{ij}}{\Lambda}$, while for multiplicative increases it is at most $\sum_i \frac{y_i}{d\Lambda}$. For the multiplicative increase to happen we must have $y_i < x_{ij} + z_{ij} < 2x_{ij}$, so the multiplicative increase is at most $\frac{2 \sum_i x_{ij}}{d\Lambda}$. Since each i appears in only one of the two summations, the total increase in cost is at most $\frac{2}{\Lambda}$ for each reinforcement step. The number of reinforcement steps is bounded in Lemma 2 along with Lemma 3 (note that we normalize Λ as in Lemma 3). Combining these gives the result. □

We can normalize B to whatever value we like. Given the expressions for the competitive ratio on the load and the cost, it is natural to set $B = \Theta(m)$. This will guarantee a competitive ratio of $O(d \log nm)$ on the cost, and a competitive ratio of $O(\log(md) \log(nm))$ on the load.

2.3 Rounding

We now show how to round our fractional solution to an integral solution. Our integral solution is $O(\log(nm) \log(md))$-competitive on the load with high probability and $O(d \log^2 nm)$-competitive on cost. Suppose we have some online fractional algorithm which guarantees that the values of x_{ij} and y_i never decrease, and maintains inequalities from the linear program in Section 2.1 except that it relaxes the first inequality to $0 \leq y_i \leq \rho_\Lambda$ and the last inequality to $\sum_{i=1}^{m} y_i c_i \leq B\rho_B$. In fact, our fractional algorithm also guarantees that $x_{ij} \leq 1$. We will show that this can be rounded in an online manner to produce integral $\hat{x}_{ij}$ and $\hat{y}_i$. Observe that our algorithm from Section 2.1 will satisfy the constraints with $\rho_\Lambda = O(\log(nm) \log(md))$ and $\rho_B = O(d \log nm)$.

Our rounding procedure is as follows. For each machine i, we compute a uniformly random $r_i \in [0, 1]$. We set $\hat{y}_i \leftarrow 1$ as soon as $y_i \log 2nm \geq r_i$. We define $M(j)$ as the set of machines with $\hat{y}_i = 1$ immediately after job j arrives. For each job j, let $y_i(j) = \min\{y_i, 1\}$ immediately after job j arrives. We observe that $x_{ij} \leq y_i(j)$, since the fourth linear program equation must hold at all times and $x_{ij} \leq 1$. We define $s_j = \sum_{i \in M(j)} \frac{x_{ij}}{y_i(j)}$. If $s_j < \frac{1}{2}$, then we immediately set $\hat{y}_i \leftarrow 1$ for all machines i and recompute s_j. We select exactly one machine i from $M(j)$ to assign j, setting $\hat{x}_{ij} \leftarrow 1$ for this machine only. Each machine is selected with probability $\frac{x_{ij}}{y_i(j) s_j}$.

Lemma 4. *The probability that we ever have $s_j < \frac{1}{2}$ after any j arrives is at most $\frac{1}{m}$; thus the increase in the expected total cost of the solution due to this case is at most B.*

Proof. Let $A(j)$ be the set of machines for which $y_i(j) \geq \frac{1}{\log 2nm}$. Clearly $A(j) \subseteq M(j)$ since all of these machines are active with probability one. Observe that if $\sum_{i \in A(j)} x_{ij} \geq \frac{1}{2}$, then $s_j \geq \frac{1}{2}$ with probability 1. Hence, we consider the case when $\sum_{i \in A(j)} x_{ij} < \frac{1}{2}$. Since $\sum_i x_{ij} \geq 1$, it follows that $\sum_{i \notin A(j)} x_{ij} > \frac{1}{2}$. The actual value of s_j will depend on the random choices of r_i, since s_j is computed by summing over only the active machines. We can write the equation $s_j \geq \sum_{i \notin A(j)} \frac{x_{ij}}{y_i(j)} \hat{y}_i$. Since $i \notin A(j)$, we can guarantee $E[\hat{y}_i] = y_i(j) \log 2nm$, implying $E[s_j] \geq \frac{1}{2} \log 2nm$.

The value of s_j is a sum of independent bernoulli variables, each of which has value at most 1 (since $x_{ij} \leq y_i(j)$). Even though the variables range in between 0 and 1, we can still apply Chernoff type bounds to conclude: $P\left[s_j < \frac{1}{2}\right] \leq P\left[s_j < (1 - \frac{1}{2}) E[s_j]\right] \leq \left(\frac{e^{-.5}}{.5^{.5}}\right)^{E[s_j]} \leq \sqrt{\frac{2}{e}}^{\frac{1}{2} \log 2nm} \leq \frac{1}{nm}$ (assuming the base of the logarithm is a sufficiently small constant). Applying the union bound, we can sum this over all j and conclude that the probability of ever having $s_j < \frac{1}{2}$ for any j is less than $\frac{1}{m}$. Hence, the increase in expected total cost due to this case is at most $\frac{1}{m} \sum_i c_i \leq B$. □

The proof of the following lemma can be found in Appendix A.2.

Lemma 5. *Every job is assigned to exactly one active machine. The expected total cost of the solution is bounded by $E[\sum_i \hat{y}_i c_i] \leq B\rho_B \log 2nm$.*

Lemma 6. *For any active machine i with $\hat{y}_i = 1$ and dimension k, with high probability the total load of $\sum_j \hat{x}_{ij} p_{ij}^k$ is at most $\Lambda[2\rho_\Lambda + 4 \log m + \beta \log(md)]$ where β is a suitably chosen constant.*

Proof. Consider active machine i. Each job j is assigned to this machine with probability $\frac{x_{ij}}{y_i(j) s_j}$. The total load of the machine on dimension k is $\sum_j \hat{x}_{ij} p_{ij}^k$; we will assume that $p_{ij}^k \leq \Lambda$ since

otherwise $x_{ij} = 0$ and thus $\hat{x}_{ij} = 0$. The problem here is that the $y_i(j)$ values change over time; if we could replace them all with the final y_i then we could use the linear program inequalities to conclude that the expected load is at most $\rho_\Lambda \Lambda$ and then apply Chernoff bounds.

Instead, we define phase α to consist of those times when $\frac{2^\alpha}{m} \le y_i(j) < \frac{2^{\alpha+1}}{m}$, for each $0 \le \alpha \le \log m$. Even though $y_i > 1$ is possible, we will never have $y_i(j) > 1$ so every j arrives in some phase. Let $J(\alpha)$ be the jobs which arrive during phase α. For $\alpha < \log m$ we have: $E[\sum_{j \in J(\alpha)} \hat{x}_{ij} p_{ij}^k] \le \sum_{j \in J(\alpha)} \frac{x_{ij}}{y_i(j) s_j} p_{ij}^k \le \sum_{j \in J(\alpha)} 2 x_{ij} p_{ij}^k \frac{m}{2^\alpha}$. However, we know that $\sum_{j \in J(\alpha)} x_{ij} p_{ij}^k \le \Lambda \frac{2^{\alpha+1}}{m}$ for any phase except the last, so we can apply this inequality to conclude $E[\sum_{j \in J(\alpha)} \hat{x}_{ij} p_{ij}^k] \le 4\Lambda$. Thus the expected total load is at most $4\Lambda \log m$ from all phases but the last. For the last phase, the expected load is at most $2\rho_\Lambda \Lambda$, since $y_i(j) = 1$ throughout. We observe that the loads are sums of Bernoulli variables with values at most Λ, so we can apply Chernoff bounds to show that with high probability the load will not exceed its mean plus $\beta \Lambda \log(md)$ on any dimension of any machine. □

In Appendix A.3, we describe how our approach combined with the rounding approach of the Generalized Assignment Problem [25] also gives an offline result similar to [16] with a substantially simpler rounding scheme. We also show that no deterministic algorithm can be competitive in Appendix A.3.

3 Machine Activation Variants

We study four versions of online load balancing with activation costs. Due to space constraints, we give some proofs in the appendix. For the version where both Λ and B are given, Section 2 gives an algorithm that is $O(\log(nm)\log(md))$-competitive on the load and $O(d\log^2(nm))$-competitive on cost. Theorems 4 and 5 are proved in Appendix A.4. Theorems 6, 7, 8, and 9 are proved in Appendices A.5, A.6, A.7, and A.8 (respectively). The logarithmic dependence on the optimum load (or budget) in some of the results is undesirable, and we observe that it would not occur in the offline setting (where we can discard the solution of previous phases and start over with each new phase). However, we will prove that this dependence is necessary for a fully online algorithm.

Theorem 4. *For the version where we are given a budget B and asked to minimize load Λ, we can produce a solution which spends at most $B\rho_B$ and has competitive ratio ρ_Λ on the load against the optimum offline with budget B. Here $\rho_\Lambda = O(\log(md)\log(nm))$ but $\rho_B = O(d\log^2(nm)\log\Lambda^*)$ where Λ^* is the optimum load (assuming we have a lower bound on the load of one; otherwise it is the ratio of maximum to minimum possible nonzero load).*

Theorem 5. *For the version where we are given load Λ and asked to minimize the budget B, we can produce a solution with load at most $\rho_\Lambda\Lambda$ which has competitive ratio ρ_B against the optimum offline which does not exceed load Λ. Here $\rho_B = O(d\log^2 nm)$ but $\rho_\Lambda = O(\log(md)\log(nm)\log B^*)$ where B^* is the ratio of the optimum budget to the minimum non-zero cost of a machine.*

Theorem 6. *Consider the version where we are given a budget B and asked to minimize load Λ. Suppose that the actual optimum load is Λ^*, and we are to guarantee that we spend a budget at most $\rho_B B$ and obtain load at most ρ_Λ times optimum. Then $\rho_B = \Omega(\min\{\frac{\log\Lambda^*}{\log\log\Lambda^* + \log\rho_\Lambda}, m\})$.*

Theorem 7. *If a deterministic algorithm guarantees to be ρ_Λ-competitive on the makespan while using at most $\rho_B B$ budget, then $\rho_B = \Omega\left(\frac{\log\Lambda^*(\log n - \log\log m - \log\log\Lambda^*)(\log m - \log\log\Lambda^*)}{(\log\rho_\Lambda + \log\log n + \log\log m + \log\log\Lambda^*)(\log\log n + \log\log m)}\right)$.*

Theorem 8. *Consider the version where we are given a load Λ and asked to minimize the cost B. Suppose that the optimum cost is B^* to stay within load Λ. We are to guarantee load at most $\Lambda\rho_\Lambda$ and cost at most ρ_B times B^*. Then $\rho_\Lambda = \Omega(\min\{\frac{\log B^*}{\log 2\rho_B}, m\})$.*

Initialize $\Lambda \leftarrow 1$, $x \leftarrow 0$, $\ell_i^{kt} \leftarrow 0$ for all i, k, t
while jobs j arrive **do**
  Update Λ_{max} and Λ_{tot}
  if $\Lambda < \max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}$ **then**
    End phase x; let $x \leftarrow \lceil \log_2 \max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\} \rceil$, $\Lambda \leftarrow 2^x$
  Place job j on machine $s = \operatorname{argmin}_i \sum_{k=1}^{d} [a^{\ell_i^{kx} + q_j^k} - a^{\ell_i^{kx}}]$; let $\ell_s^{kx} \leftarrow \ell_s^{kx} + q_j^k$ for all k

Algorithm 2: Assign-Jobs.

Theorem 9. *There is an $O(d \log^2 nm)$-competitive algorithm for minimizing a linear combination of the cost B and load Λ, namely $c_B B + c_\Lambda \Lambda$. Here, the coefficients c_B, c_Λ are constants and $c_B B + c_\Lambda \Lambda$ is the value of the objective function on a schedule with energy cost B and makespan Λ.*

4 Vector Load Balancing

We give an $O(\log d)$-competitive algorithm to minimize the makespan, improving over the offline $O(\log^2 d)$-approximation in [8] (for arbitrary d). Note that [8] shows that even in the offline case, no constant approximation is possible unless NP = ZPP. We extend our algorithm to be simultaneously $O(\log d)$-competitive on energy, provided we know $\Lambda_{max} = \max_{k,j} p_j^k$ in advance (p_j^k is the load of job j on component k). We show that this additional information is necessary.

4.1 Minimizing the Makespan

The algorithm of [2] depends heavily on maintaining an estimate Λ of the optimum value. We use the maximum coordinate of any single job $\Lambda_{max} = \max_{k,j} p_j^k$ and the load induced by placing all jobs on one machine $\Lambda_{tot} = \max_k \sum_j p_j^k$ as lower bounds in Algorithm 2. We run in phases where for each phase, we use an estimate Λ of the optimum makespan. If Λ is too small, we adjust and start a new phase. We also make use of $a = 1 + \frac{1}{\gamma}$ for some $\gamma > 1$. Let $\ell_i^{kx}(j)$ be the load normalized by Λ on component k of machine i during phase x after all jobs up through j are assigned (we sometimes omit j if the context is clear). Let p_j^k be the load induced on dimension k by job j, and $q_j^k = p_j^k / \Lambda$. Lemma 7 is proved in Appendix B.1 (this is a modification of the proof in [2]).

Lemma 7. *In Algorithm 2, during each phase x:* $\sum_{k=1}^{d} \sum_{i=1}^{m} a^{\ell_i^{kx}} (\gamma - 1) \le \gamma m d$.

At this point, we can follow [2] and use Lemma 7 for an $O(\log dm)$ bound. We will improve our competitive ratio by exploiting the identical nature of our machines. Let $\mu^x = \max_{i,k} \ell_i^{kx}$.

Lemma 8. *For all machines i, i' and phases x, $\sum_{k=1}^{d} a^{\ell_i^{kx}(f)} \le d + \frac{\mu^x}{\gamma} \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}$ where f is the final job of phase x.*

Proof. Let J_i be the set of jobs given to machine i during phase x. If $j \in J_i$, machine i must minimize $\Delta_i(j) = \sum_{k=1}^{d} a^{\ell_i^{kx}(j-1) + q_j^k} - a^{\ell_i^{kx}(j-1)}$. Hence, for any i', we have $\sum_{k=1}^{d} (a^{\ell_i^{kx}(f)} - 1) = \sum_{j \in J_i} \Delta_i(j) \le \sum_{j \in J_i} \Delta_{i'}(j) \le \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)} \sum_{j \in J_i} (a^{q_j^k} - 1)$. Since $p_j^k \le \Lambda$, we have $0 \le q_j^k \le 1$ and thus $\gamma(a^{q_j^k} - 1) \le q_j^k$. This gives

$$\sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)} \sum_{j \in J_i} (a^{q_j^k} - 1) \le \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)} \sum_{j \in J_i} \frac{q_j^k}{\gamma} = \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)} \frac{\ell_i^{kx}(f)}{\gamma} \le \frac{\mu^x}{\gamma} \sum_{k=1}^{d} a^{\ell_{i'}^{kx}(f)}.$$

```
Initialize M ← 1
while M < m do
    Run algorithm Assign-Jobs on M new virtual machines until Λ_tot > MΛ_max
    M ← min{2M, m}
while jobs are still arriving do
    Run algorithm Assign-Jobs on all real machines ignoring previous loads
```

Algorithm 3: Power-and-Makespan.

Putting our inequalities together and rearranging terms finishes the proof. □

Theorem 10. *Algorithm 2 is $O(\log d)$-competitive on the makespan.*

Proof. Fix phase x and let s and s' be the machines with maximal and minimal $\sum_{k=1}^{d} a^{\ell_i^{kx}}$ (respectively) at the end of x. Combining Lemmas 7 and 8 gives

$$a^{\mu^x} \leq \sum_{k=1}^{d} a^{\ell_s^{kx}(f)} \leq d + \frac{\mu^x}{\gamma} \sum_{k=1}^{d} a^{\ell_{s'}^{kx}(f)} \leq d + \frac{\mu^x}{\gamma}\left(\frac{\gamma d}{\gamma - 1}\right) = d + \frac{\mu^x d}{\gamma - 1}.$$

Taking the logarithm of both sides gives $\mu^x \leq \log_a\left(\frac{d}{\gamma-1}\right) + \log_a(\mu^x + \gamma - 1)$. Thus, $\mu^x - \log_a(\mu^x + \gamma - 1) = O(\log d)$. Note that if for some constant c we have $c - \log_a(c + \gamma - 1) = \frac{c}{2}$, then for all $\mu^x \geq c$, we have $\mu^x = O(\log d)$. The makespan during phase x is $2^x O(\log d)$. For our last phase g, our total load is at most $2^{g+1}O(\log d)$. Since $g = \lceil \log(\max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}) \rceil$ and $\max\{\Lambda_{max}, \frac{1}{m}\Lambda_{tot}\}$ is a lower bound on optimum, our algorithm is $O(\log d)$-competitive. □

4.2 Simultaneously Minimizing Energy

Given the value of Λ_{max} in advance, we compress all jobs onto a small number of machines, then gradually open up more machines as our estimate of Λ_{tot} increases. Algorithm 3 does this with virtual machines. As there are at most $2m$ virtual machines in total, we identify two virtual machines with each real machine. We prove Theorem 12 in Appendix B.2. Theorem 12 establishes that advance knowledge of Λ_{max} (or some comparable advance knowledge) is necessary to have a competitive ratio independent of m.

Theorem 11. *Algorithm 3 is $O(\log d)$-competitive on both makespan and power.*

Proof. Within a call to Assign-Jobs, we guarantee that $\Lambda_{tot} \leq M\Lambda_{max}$. Thus by Theorem 10, we place a load of at most $\Lambda_{max}O(\log d)$ on any virtual machine. Then after all jobs are placed, the load of any real machine is at most the sum of the loads of two virtual machines (each at most $\Lambda_{max}O(\log d)$), plus any load placed by the last instance of Assign-Jobs when $M = m$. Thus the makespan is at most $2\Lambda_{max}O(\log d) + \frac{1}{m}\Lambda_{tot}O(\log d)$. We observe that at most $2M$ machines have non-zero load, so the total power is at most $4M\Lambda_{max}O(\log d) + 2\Lambda_{tot}O(\log d)$. Optimum power is Λ_{tot} and the algorithm guarantees $\Lambda_{tot} > \frac{M}{2}\Lambda_{max}$, which completes the proof.

□

Theorem 12. *Suppose we have an online algorithm which is α-competitive on the makespan and β-competitive on energy. Then $\beta \geq \frac{1}{2\alpha}\min(d, \log_{2\alpha} m)$.*

References

[1] N. Alon, B. Awerbuch, Y. Azar, N. Buchbinder, and J. Naor. The online set cover problem. In *Proceedings of the 35th annual ACM Symposium on Theory of Computing*, 2003.

[2] J. Aspnes, Y. Azar, A. Fiat, S. Plotkin, and O. Waarts. On-line load balancing with applications to machine scheduling and virtual circuit routing. In *Proceedings of the 25th annual ACM Symposium on Theory of Computing*, 1993.

[3] B. Awerbuch, Y. Azar, E. Grove, M. Kao, P. Krishnan, and J. Vitter. Load balancing in the l_p norm. In *Proceedings of the 36th annual IEEE Symposium on Foundations of Computer Science*, 1995.

[4] Y. Azar, U. Bhaskar, L. Fleischer, and D. Panigrahi. Online mixed packing and covering. In *Proceedings of the 24th annual ACM-SIAM Symposium on Discrete Algorithms*, 2013.

[5] N. Bansal, A. Caprara, and M. Sviridenko. Improved approximation algorithms for multidimensional bin packing problems. In *Proceedings of the 47th annual IEEE Symposium on Foundations of Computer Science*, 2006.

[6] Y. Bartal, A. Fiat, H. Karloff, and R. Vohra. New algorithms for an ancient scheduling problem. In *Proceedings of the 24th annual ACM Symposium on Theory of Computing*, 1992.

[7] N. Buchbinder and J. Naor. Fair online load balancing. In *Proceedings of the 18th annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2006.

[8] C. Chekuri and S. Khanna. On multi-dimensional packing problems. In *Proceedings of the 10th annual ACM-SIAM Symposium on Discrete Algorithms*, 1999.

[9] D. DeWitt and J. Gray. Parallel database systems: The future of high performance database systems. *Communications of the ACM*, 1992.

[10] M. Garofalakis and Y. Ioannidis. Scheduling issues in multimedia query optimization. *ACM Computing Surveys*, 1995.

[11] M. Garofalakis and Y. Ioannidis. Multi-dimensional resource scheduling for parallel queries. In *Proceedings of 1996 ACM SIGMOD International Conference on Management of Data*, 1996.

[12] M. Garofalakis and Y. Ioannidis. Parallel query scheduling and optimization with time- and space-shared resources. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[13] D. Hochbaum and D. Shmoys. Using dual approximation algorithms for scheduling problems theoretical and practical results. *Journal of the ACM*, 34(1):144–162, 1987.

[14] S. Irani and K. Pruhs. Algorithmic problems in power management. *SIGACT News*, 36(2):63–76, 2005.

[15] R. Karp, M. Luby, and A. Marchetti-Spaccamela. A probabilistic analysis of multidimensional bin packing problems. In *Proceedings of the 16th annual ACM Symposium on Theory of Computing*, 1984.

[16] S. Khuller, J. Li, and B. Saha. Energy efficient scheduling via partial shutdown. In *Proceedings of the 21st annual ACM-SIAM Symposium on Discrete Algorithms*, 2010.

[17] L. Kou and G. Markowsky. Multidimensional bin packing algorithms. *IBM Journal of Research and Development*, 1977.

[18] R. Lavi and C. Swamy. Truthful mechanism design for multi-dimensional scheduling via cycle monotonicity. In *Proceedings of the 8th annual ACM conference on Electronic Commerce*, 2007.

[19] J. Li and S. Khuller. Generalized machine activation problems. In *Proceedings of the 22nd annual ACM-SIAM Symposium on Discrete Algorithms*, 2011.

[20] S. Lo, R. Chen, and Y. Huang. Multi-constraint system scheduling using dynamic and delay ant colony system. *New Trends in Applied Artificial Intelligence*, 4570, 2007.

[21] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": temperature-aware workload placement in data centers. In *Proceedings of the USENIX annual Technical Conference*, 2005.

[22] K. Pruhs, J. Sgall, and E. Torng. Online scheduling. *Handbook of scheduling: algorithms, models, and performance analysis*, 2004.

[23] M. Rofouei, T. Stathopoulos, S. Ryffel, W. Kaiser, and M. Sarrafzadeh. Energy aware high performance computing with graphical processing units. In *Proceedings of USENIX 1st Workshop on Power Aware Computing Systems*, 2008.

[24] S. Ryffel, T. Stathopoulos, D. McIntire, W. Kaiser, and L. Thiele. Accurate energy attribution and accounting for multi-core systems. In *Technical Report 67, Center for Embedded Network Sensing*, 2009.

[25] D. Shmoys and É. Tardos. An approximation algorithm for the generalized assignment problem. *Math. Program.*, 62(3):461–474, 1993.

[26] T. Stathopoulos, D. McIntire, and W. Kaiser. The energy endoscope: Real-time detailed energy accounting for wireless sensor nodes. In *Proceedings of the 7th annual International Conference on Information Processing in Sensor Networks*, 2008.

A Machine Activation Problem

A.1 Proof of Theorem 1

Inequality 3 follows from the termination condition of step three of the algorithm. The other two inequalities hold initially since all variables are zero. When j arrives, we set $x_{ij} \leftarrow \frac{1}{jm}$ for every i where $q_{ij}^k \leq 1$ along each dimension k, but we also increase all y_i by $\frac{1}{jm}$, so both inequalities will continue to hold.

Later, we might increase x_{ij} by some z_{ij}. However, we will only do this if $x_{ij} + z_{ij} \leq y_i$, so $x_{ij} \leq y_i$ still holds. When we increase x_{ij} in this way, we will also increase y_i by $z_{ij} \max_k q_{ij}^k$, which guarantees that $\sum_j x_{ij} p_{ij}^k \leq \Lambda y_i$ will continue to hold for all i and all k.

A.2 Proof of Lemma 5

. The rounding scheme assigns each job to exactly one active machine, and the set of active machines only grows over time as the y_i values are non-decreasing. The manner in which the $\hat{y}_i$ are determined along with the inequality $\sum_i y_i c_i \leq B\rho_B$ produce the cost bound (since $E[\hat{y}_i] \leq y_i \log 2nm$). Note that the additional expected cost as specified in Lemma 4 in the case that $s_j < \frac{1}{2}$ is negligible and does not change the competitive ratio asymptotically.

A.3 Machine Activation Offline Scheme and Lower Bound

For the offline case, we can simply solve the linear program, so $\rho_\Lambda = \rho_B = 1$. Instead of rounding up when $y_i \log 2nm \geq r_i$, we can round up when $y_i \log 2n \geq r_i$. This increases the probability of the bad event where some $s_j < \frac{1}{2}$ to be a small constant instead of $\frac{1}{m}$, but we can simply discard our solution and retry whenever this occurs. This means we can get a solution of cost $O(\log n)$ times B, such that a fractional solution exists which exceeds Λ by at most a constant factor on any machine. We can then convert our fractional solution to an integer solution using the rounding approach of the Generalized Assignment Problem [25]. This attains roughly the same bounds as [16] for the offline case.

There is a simple example which indicates that no deterministic approach to this problem can succeed. We simply give a series of requests each of which can run on every machine *except the ones where the preceding requests were scheduled.* This forces a deterministic algorithm to pay for all m machines, whereas the offline optimum could activate only a single machine. The Online Set Cover result of Alon et al. [1] got around this by presuming that we know in advance the set of elements (jobs) which *might* be requested in future, and allowed the competitive ratio to depend on the size of this set. This approach seems less reasonable for our problem than theirs, but in any case a modification of their derandomization should work in the same model.

A.4 Proof of Theorems 4 and 5

We now consider the version where one of B, Λ is given up front, and we are asked to minimize the other. Our results for these two theorems are obtained using standard doubling techniques in combination with running our online, randomized algorithm from Section 2.3.

In particular, a natural approach to this version is to guess the value of the parameter we are trying to optimize, and then run the online algorithm until it becomes apparent that this value is too small. We then double the parameter and continue. The algorithm therefore runs in a number of phases, and the total budget will be the sum of the budgets spent at each phase (similarly the total load is the sum of loads generated at each phase). For the parameter we are trying to optimize,

this sum is geometric and therefore increases the end value by only a constant; however, the other parameter increases by a factor of the number of phases. This gives the two theorems.

A.5 Proof of Theorem 6

We are given m machines, each of which has activation cost $c_i = 1$. Our budget is $B = 1$, and we are asked to minimize the load online. The adversary submits up to n jobs, where $n = \min\{\frac{\log \Lambda^*}{\log(2\rho_\Lambda \log \Lambda^*)}, m\}$. Each job j can run only on machines j through m. To run job j on machine i, we will induce a load $p_{ij} = L^{i-1}$. We will define $L = 2\rho_\Lambda \log \Lambda^*$. Note that $L^n \leq \Lambda^*$ and $n < L$.

We observe that the offline solution can handle jobs 1 through j using only machine j while staying within its budget, and the total load in this case will be $jL^{j-1} \leq nL^{n-1} < L^n \leq \Lambda^*$. If our algorithm were to place any job on machine $j+1$ or higher with probability more than half, then our expected load would be at least $\frac{1}{2}L^j > \rho_\Lambda j L^{j-1}$, violating our competitive ratio on the load. Thus our algorithm places job j on machine j with probability at least half, implying that we activate machine j with probability at least half. This applies to the first n machines, so our algorithm pays a cost of at least $\frac{1}{2}n$ in expectation, implying that $\rho_B = \Omega(\min\{\frac{\log \Lambda^*}{\log\log \Lambda^* + \log \rho_\Lambda}, m\})$.

A.6 Proof of Theorem 7

We first consider a variant of the problem where a universe U of n jobs is pre-announced, but only some subset of $n' \leq n$ jobs arrive. It is clear that this problem is no harder than the original problem since this is only providing more information to the online algorithm (and n only gets larger). Hence, a lower bound for this variant provides a lower bound for the original version. Let OPT denote the optimal solution and let ALG denote the deterministic algorithm. Let p, q, r be positive integers. Our universe $U = \{0, 1, 2, \ldots, n-1\}$ where $n = 2^p pq^2 r$. We will partition U into r "phases" and each phase into pq^2 "blocks". The first phase will contain precisely the first $2^p pq^2$ jobs, the second phase will contain the next $2^p pq^2$ jobs and so on. Within each phase k, the first block X_1^k contains the first 2^p jobs of the phase, the second block X_2^k contains the second 2^p jobs of the phase and so on up to block $X_{pq^2}^k$ which contains the last 2^p jobs of the phase.

For each set $B = \{b_1, b_2, \ldots, b_q\} \subseteq \{1, 2, \ldots, pq^2\}$ with $b_1 < b_2 < \cdots < b_q$ and each set $I = \{i_1, i_2, \ldots, i_q\}$ where $1 \leq i_t \leq p$ for all t and each $1 \leq k \leq r$, we will have a machine $M_{B,I}^k$. This machine will be able to run any job from phases $1, 2, \ldots, k-1$, and any job in $X_{b_t}^k$ that has the i_t-th least significant bit in its binary encoding set to 1. Each of these jobs will induce load $\frac{1}{pqk}L^k$ on machine $M_{B,I}^k$ for some $L > \rho_\Lambda pq(r+1)$. Each machine will have activation cost 1 and our total allowed budget will be 1. We note that the number of machines is $m = \binom{pq^2}{q} p^q r$.

The instance will work in phases. In each phase k, exactly pq jobs will arrive. We will ensure that at any time OPT can place all jobs (including those from previous phases) on a single machine within the current phase. Since in phase k a total of pqk jobs will have arrived, this means the OPT makespan is at most L^k at the end of phase k.

Jobs arrive within phase k as follows: Take the job from X_1^k that has all p least significant bits set to 1. ALG must select one machine on which to place this job. This job cannot go on any machine in a previous phase. Moreover, if ALG places the job on a machine in a future phase then this will induce load:

$$\frac{1}{pq(k+1)}L^{k+1} > \frac{1}{pq(k+1)}(\rho_\Lambda pq(r+1))L^k \geq \rho_\Lambda L^k.$$

Thus, since ALG guarantees to be ρ_Λ-competitive on makespan, this cannot happen. So it follows that ALG must place this job on some machine in the current phase. This machine is able to do

all jobs in X_1^k with some bit b_1 set to 1. Thus, the next job to arrive will be the job from X_1^k that has all p least significant bits except b_1 set to 1. Clearly, ALG must now choose a different machine to process this job. This continues for k steps at which point, ALG has activated k machines, but OPT could have activated only one.

At this point, ALG has k different machines which can accommodate jobs from at most pq blocks in phase k. Since there are pq^2 blocks, there must be another block we can select and repeat the above process. In fact, we can repeat this q times forcing ALG to activate pq machines. Notice OPT can select the single machine that handles elements from exactly the blocks we select (and exactly the correct bit in each block). In total, ALG will be forced to activate pqr machines whereas OPT is able to activate only one machine.

Notice that $p < \log n$, $q < \log m$ and $r+1 < \log \Lambda^*$. Thus we can set $L = \rho_\Lambda \log n \log m \log \Lambda^* > \rho_\Lambda pq(r+1)$, to get that $r = \frac{\log \Lambda^*}{\log \rho_\lambda + \log\log n + \log\log m + \log\log \Lambda^*}$. Since $n = 2^p pq^2 r$ we have $p \geq \Omega(\log n - \log\log m - \log\log \Lambda^*)$. Finally, since $m \leq (pq^2)^q p^q r$ we have $q \geq \frac{\log m - \log\log \Lambda^*}{\log\log n + \log\log m}$. It follows that

$$\rho_B \geq \Omega\left(\frac{\log \Lambda^*(\log n - \log\log m - \log\log \Lambda^*)(\log m - \log\log \Lambda^*)}{(\log \rho_\Lambda + \log\log n + \log\log m + \log\log \Lambda^*)(\log\log n + \log\log m)}\right)$$

A.7 Proof of Theorem 8

We are given m machines labeled $i = 0$ through $i = m-1$, where machine i has an activation cost of $c_i = (2\rho_B)^i$. We are asked to maintain a load of at most $\Lambda = 1$. The adversary gives at most n requests, where $n = \min\{\frac{\log B^*}{\log 2\rho_B}, m\}$. Request $j \geq 1$ has $p_{ij} = 1$ for machines $i = 0$ and $i = j$ and otherwise has infinite load. We observe that the optimum can obey the load constraints on the first k jobs by activating machines 0 through $k-1$, placing job k on machine 0 and job $j < k$ on machine j. This has cost $\sum_{i=0}^{k-1}(2\rho_B)^i < 2(2\rho_B)^{k-1} < B^*$. Thus the online algorithm cannot activate any machine k or higher without violating the ρ_B bound on the competitive ratio, from which it follows that job k must be placed on machine 0. Yet this applies to every job, so we must place all n jobs on machine 0, violating the load by $\rho_\Lambda = \Omega(\min\{\frac{\log B^*}{\log 2\rho_B}, m\})$.

A.8 Proof of Theorem 9

We will run in phases, where in phase i we assume that $B = \frac{2^i}{c_B}$ and $\Lambda = \frac{2^i}{c_\Lambda}$, and run the online algorithm of Section 2 for each phase.

Suppose that the actual optimum value is $2^{\alpha-1} < OPT \leq 2^\alpha$. Thus we have $B^* \leq \frac{2^\alpha}{c_B}$ and $\Lambda^* \leq \frac{2^\alpha}{c_\Lambda}$, so the algorithm will terminate on phase α or earlier.

The total load generated by the algorithm is the sum of loads from each phase up to α. The load from phase i is at most $(\log nm)(\log md)2^i$, and summing these gives $\Lambda \leq O(2^{\alpha+1}(\log nm)(\log md))$. The total cost can be bounded by $B \leq O(2^{\alpha+1}d(\log nm)^2)$. Combining these gives the result.

B Vector Load Balancing

B.1 Proof of Lemma 7

Let $\lambda^k(j) = \frac{1}{m}\sum_{t=1}^{j} q_t^k$. We define the potential function $\Phi(j) = \sum_{k=1}^{d}\sum_{i=1}^{m} a^{\ell_i^{kx}(j)}(\gamma - \lambda^k(j))$.

Note that at the beginning of phase x, all the values $\ell_i^{kx}(j)$ are zero, so the value of the potential is at most γmd. We claim that as the phase continues, this potential function can only decrease. Observing that throughout the phase $\lambda^k(j) \leq 1$ (since otherwise we would have $\frac{1}{m}\Lambda_{tot} > \Lambda$ and the

phase would end) will then establish the lemma. Now suppose that job j during phase x is placed upon machine s. The potential function will change as follows:

$$\begin{aligned}
\Phi(j) - \Phi(j-1) &= \sum_{k=1}^{d}(\gamma - \lambda^k(j-1))(a^{\ell_s^{kx}(j)} - a^{\ell_s^{kx}(j-1)}) - \sum_{k=1}^{d}\sum_{i=1}^{m}(\lambda^k(j) - \lambda^k(j-1))(a^{\ell_i^{kx}(j)}) \\
&\leq \sum_{k=1}^{d}\gamma(a^{\ell_s^{kx}(j)} - a^{\ell_s^{kx}(j-1)}) - \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m} q_j^k(a^{\ell_i^{kx}(j)}) \\
&\leq \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m}\gamma(a^{\ell_i^{kx}(j-1)+q_j^k} - a^{\ell_i^{kx}(j-1)}) - q_j^k(a^{\ell_i^{kx}(j)}) \\
&\leq \frac{1}{m}\sum_{k=1}^{d}\sum_{i=1}^{m} a^{\ell_i^{kx}(j-1)}(\gamma a^{q_j^k} - \gamma - q_j^k) \\
&\leq 0
\end{aligned}$$

The first inequality follows from the definition of λ^k and the observation that $\lambda^k(j-1) \leq \Lambda$. The second from our choice of machine s. The third inequality comes from the observation that for any i, $a^{\ell_i^{kx}(j)} \geq a^{\ell_i^{kx}(j-1)}$. The last inequality comes from the definition of a, γ; in particular the fact that for any $y \in [0,1]$ we have $\gamma(a^y - 1) \leq y$ along with the fact that $q_j^k \in [0,1]$ because $\Lambda \geq \Lambda_{max} \geq p_j^k$. Since the potential can never increase, we conclude that the *final* potential is at most the *initial* potential of γmd.

B.2 Proof of Theorem 12

Jobs will arrive in $p = \min(d, \log_{2\alpha} m)$ phases. During phase i, we receive $\frac{m}{(2\alpha)^{i-1}}$ vectors each with $(2\alpha)^{i-1}$ in the i'th coordinate and zero elsewhere. After i phases, since vectors from different phases place load on different coordinates, the optimum makespan of $(2\alpha)^{i-1}$ can be achieved by assigning all jobs from any given phase to different machines. Then the algorithm must guarantee makespan at most $\alpha(2\alpha)^{i-1}$ at the end of phase i. It follows that we cannot place more than α of the phase i job vectors on the same machine, and thus that at least $m_i \geq \frac{m}{\alpha(2\alpha)^{i-1}}$ machines received at least one phase i job.

The optimum energy is m achieved by scheduling all jobs on one machine. However, for the algorithm to be β-competitive on energy we need

$$\beta m \geq \sum_i m_i[(2\alpha)^{i-1} - (2\alpha)^{i-2}] \geq m\sum_i \frac{1}{2\alpha} \geq \frac{mp}{2\alpha}.$$