On Energy-aware Allocation and Execution for Batch and Interactive MapReduce

Yijun Ying
EPFL
Station 14
Lausanne, Switzerland
yijun.ying@epfl.ch

Robert Birke
IBM Research-Zurich Lab
Säumerstrasse 4
Rüschlikon, Switzerland
bir@zurich.ibm.com

Cheng Wang
Penn State University
University Park
PA, 16802, USA
cxw967@cse.psu.edu

Lydia Y. Chen
IBM Research-Zurich Lab
Säumerstrasse 4
Rüschlikon, Switzerland
yic@zurich.ibm.com

Gautam Natarajan
Texas A&M University
College Station
TX, 77843, USA
gautam@tamu.edu

ABSTRACT

The energy-performance optimization of datacenters becomes ever challenging, due to heterogeneous workloads featuring different performance constraints. In addition to conventional web service, MapReduce presents another important workload class, whose performance highly depends on data availability/locality and shows different degrees of delay sensitivities, such as batch vs. interactive MapReduce. However, current energy optimization solutions are mainly designed for a subset of these workloads and their key features. Here, we present an energy minimization framework, in particular, a concave minimization problem, that specifically considers time variability, data locality, and delay sensitivity of web, batch-, and interactive-MapReduce. We aim to maximize the usage of MapReduce servers by using their spare capacity to run non-MapReduce workloads, while controlling the workload delays through the execution of MapReduce tasks, in particular batch ones. We develop an optimal algorithm with complexity $O(T^2)$ in case of perfect workload information, T being the length of the time horizon in number of control windows, and derive the structure of optimal policy for the case of uncertain workload information. Using extensive simulation results, we show that the proposed methodology can efficiently minimize the datacenter energy cost while fulfilling the delay constraints of workloads.

1. INTRODUCTION

Datacenters are standard IT (Information Technology) platforms, which consume a significant amount of energy to host a wide variety of conventional and emerging workloads, such as web services vs. MapReduce, featuring different performance requirements and workload characteristics. Typically, web services interact with clients, who require stringent response times and thus real time processing. To guarantee the throughput of large-scale data processing, MapReduce/Hadoop [1] is a simple yet powerful framework to process large amounts of data chunks organized in distributed file systems, e.g., Hadoop Distributed File Systems (HDFS). Moreover, with the recent adoption of MapReduce alongside real time queries [6,9], MapReduce workloads evolve from traditional throughput sensitive batch jobs to increasingly delay sensitive interactive jobs, such as Sparks [20] on stream processing, Natjam [8] on batch, and BlinkDB [4] on interactive queries. Energy-aware resource allocation for such a diverse set of workloads is thus ever challenging, and unfortunately existing solutions are often designed for a subset of these three workload types.

As web and service workloads show strong time-varying behaviors, dynamic sizing of the datacenter [13] by controlling the number of active servers is shown effective to minimize energy. To better utilize the equipped capacity and benefit from time-varying power supply, consolidating and delaying the execution of data intensive applications during web workload low load periods can further harvest significant cost savings for datacenters [14, 17]. However, the issues related to data locality of MapReduce workloads are unfortunately often overlooked or over simplified.

To simultaneously address data availability and energy minimization, dedicated MapReduce clusters leverage two types of control knobs independently, i.e., the fraction of on-times of the entire cluster and the fraction of on-servers at each time period. On one hand, clusters delay the execution of batch MapReduce jobs, such as Google index computation [3] or bank interest calculation, to process them on the *entire cluster* [11], depending on the energy price or other workload demands. As such, the maximum degree of data locality is guaranteed minimizing execution time and energy consumption. On the other hand, motivated by the practice of duplicating data chunks (usually by a factor of three [2]), a few solutions modify the underlying file system so that a fraction of servers, e.g., one third of the servers [12], are always available and contain one copy of all the data chunks. Nonetheless, unavoidable delay can incur in both solutions and this is not acceptable for interactive MapReduce. Beamer [6], specially designed for interactive MapReduce systems, shows promising energy savings by serving interactive MapReduce on a subset of servers all the time and batching MapReduce on the entire cluster once a day. However, it is not clear how one can dynamically execute(delay) the MapReduce workloads on allocated servers so to achieve the optimal tradeoff between data locality and energy efficiency.

In this paper, we address the question how to minimize energy consumption of executing web applications, batch MapReduce, and interactive MapReduce, considering their distinct workload characteristics, i.e., time-variability and data locality, and different performance requirements, i.e., throughput vs. delay. The system of interest is composed of web servers and dedicated MapReduce

Copyright is held by author/owner(s).

servers where a distributed file system is deployed. As our aim is to design a non-intrusive solution, i.e., not to modify the underlying file systems, we propose to keep all MapReduce servers on at all times so that data availability is ensured. In order to minimize the energy consumption of the entire system, i.e., total number of on servers, we try to execute all three types of workloads on MapReduce servers only as to size the web cluster as small as possible. In particular, we consider two control variables over discrete windows: delaying the execution of batch MapReduce and allocating a fraction of MapReduce servers for batch and interactive workloads. We employ dynamic programming to derive the optimal decisions, and simulation to evaluate the proposed solutions under various workload scenarios.

Formally, we formulate an energy minimization problem over a discrete horizon, constrained on different degrees of delay in batch and interactive MapReduce – a concave minimization problem. The specific control variables are the number of servers for MapReduce workloads and the amount of batch MapReduce per control window, which thus specifies the amount of batch jobs to be delayed. Assuming perfect knowledge on all three types of workloads, we develop an algorithm, which can efficiently achieve the optimal solution with a complexity of $O(T^2)$, where T is the number of control windows. Finally, we build an event driven simulator to evaluate the proposed algorithm under different workload scenarios, in comparisons with simple algorithms that overlook the data locality and delay sensitivity of batch and interactive MapReduce.

Our specific contributions can be summarized in the following. To minimize the energy cost of executing delay and throughput sensitive applications, we consider important tradeoffs among crucial parameters, i.e., data locality, time-variability, and delay sensitivity, of web applications, batch MapReduce and interactive MapReduce. We are able to dynamically and optimally determine the fraction of batch MapReduce workloads to be delayed by allocation of number of MapReduce servers, via analytical constructions, as well as, event driven simulations under various workload scenarios.

The outline of this work is as follows. Section 2 provides an overview of the system and a formal definition of the problem statement. The algorithm for perfect workload information is detailed in Section 3. Section 4 presents the experimental set up and simulations results. Section 5 compares related work, followed by summaries and conclusions in Section 6.

2. SYSTEM AND PROBLEM STATEMENT

In this section, we first describe the system and assumptions considered by this study and formally present the problem statement.

2.1 System

The system hosts three types of workloads: web applications, batch MapReduce, and interactive MapReduce, characterized by different degrees of data locality, time variability, and delay sensitivity. In terms of data locality, both types of MapReduce workloads require data access to the distributed file system hosted across the MapReduce servers. Web applications and interactive MapReduce are very sensitive to delay and have priority to be executed immediately with sufficient server capacity. On the contrary, batch MapReduce is latency insensitive and its execution can be delayed. As for time variability, web service workload types are known to have regular time varying patterns.

In this paper we consider a time slotted system model with each control window of length τ. The decisions are made at the beginning of each control window. We use $\lambda_{i,t}$ to denote the task arrival rates of type $i \in \{w,b,c\}$ workloads at time period t, where w,b,c represent web, batch MapReduce, and interactive MapReduce. We assume task arrival rates of both MapReduce are drawn from geometric distributions. Moreover, we assume the average execution rate per task is $\mu_i, i \in \{w,b,c\}$. As for the performance constraints, web workloads need to satisfy certain response time targets, where batch and interactive MapReduce tasks just need to be completed within certain periods of time, such as a day vs. control window.

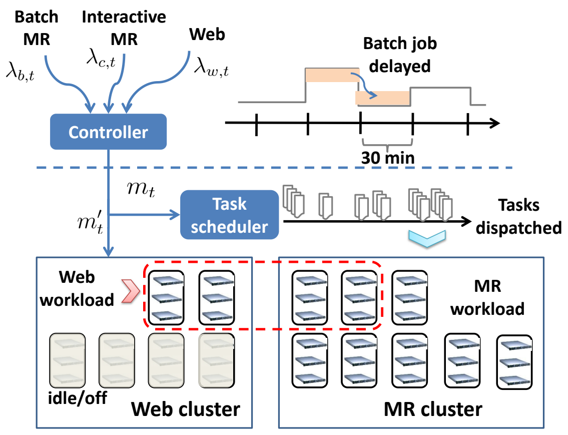


Figure 1: System schematic. On the high level, the controller takes as input the workload characteristics and outputs dynamic server allocation decisions for the three types of workloads; the scheduler then dispatches tasks according to different scheduling policies onto the lower level.

As web workloads have no dependency on the file system, we assume web workloads can be executed on both web and MapReduce servers. We further assume that the interferences between CPU-intensive web applications and IO-intensive HDFS are negligible. Therefore, the number of active web servers can be dynamically sized depending on the workload demands. On the contrary, to ensure 100% data availability and achieve energy saving, we propose that all MapReduce servers are kept *on* and allocated to the three types of workloads, when deemed appropriate. Moreover, due to the concern of interferences between web and MapReduce workloads [21, 22], we do not co-execute on the same server. Essentially, there are m_t and m'_t number of servers dedicated for MapReduce and web workloads, respectively. Among m_t servers, interactive MapReduce tasks have higher priority over batch MapReduce tasks. The decisions of m_t and m'_t are based on the workload demands and performance constraints. Consequently, the total number of active servers is $\max\{m_t + m'_t, M\}$. One can thus write the energy cost for the entire cluster for control windows $\{1 \dots t \dots T\}$ as

$$\sum_{t=1}^{T} K \cdot \max\{m_t + m'_t, M\}, \tag{1}$$

where M is the total number of servers in MapReduce cluster, T the time horizon and K the unit energy cost per *on/active* server. Note that we assume the energy cost of *off/inactive* servers to be zero.

Scheduler and Controller. When different tasks arrive at the system, they are immediately sent to the scheduler, which can employ different scheduling policies using different queue structures. Tasks are then scheduled on servers, according to their types. Another important system component is the controller, which implements the server allocation algorithm across the three types of workloads in discrete control windows. The high level schematic is summarized in Figure 1.

2.2 Assumption on Data Locality

Data locality defines that a task can find a copy of data on the local execution machine instead of getting the data from a remote machine. Denote the average execution time of a task using local copy by $1/\mu$. The execution time of using remote copy is much higher, here, assumed by a slowdown factor of α, i.e., $\frac{1}{\mu} * \alpha$. Note that we assume batch and interactive MapReduce tasks have the same average execution time, i.e., $\mu_b = \mu_c = \mu$, since in MapReduce, large jobs will be divided into small tasks and each task will deal with a constant amount of data, e.g., 64 MB by default in Hadoop.

To estimate the throughput of MapReduce, it is very important to know the probability of tasks being executed with local data, denoted as P_l.

Following the common practice of data replication in MapReduce clusters, we assume a data chunk to have γ replica, $1 \leq \gamma \leq M$. Given an allocation of m MapReduce servers, the probability of finding at least one local copy within the m servers can be computed as one minus the probability that no local copy is found, i.e.:

$$P_l(m) = 1 - \frac{\binom{M-\gamma}{m}}{\binom{M}{m}} = 1 - \prod_{i=0}^{\gamma-1} \frac{M-m-i}{M-i}.$$

$Pl(m)$ is always equal to 1 when $\gamma > Mm$, because one can find at least one replica among any m number of servers. As a function of m, $P_l(m)$ can change in every control window, as the number of allocated MapReduce servers, m_t, changes in time with workload demands. We further note here that $P_l(m)$ is an upper bound, assuming the underlying scheduler is always able to schedule the task on the server having a local copy.

Assume each control window to have length τ, one can estimate the MapReduce throughput (in units of tasks per control window) of a server being

$$X(m) = \frac{\tau}{\frac{P_l(m)}{\mu} + \frac{1-P_l(m)}{\mu/\alpha}} . \qquad (2)$$

Note that as $P_l(m)$ is an upper bound, the throughput presented here is also an optimal case, assuming optimal scheduling.

Putting it all together, the MapReduce optimal throughput of the entire system is $m \cdot X(m)$. We provide a numerical example to illustrate how MapReduce throughput changes with the number of allocated MapReduce servers under different replication factors γ in Figure 2, where the cluster has 1000 nodes. From the figure one can see that the throughput of one server, i.e., $X(m)$, is increasing in m, since the data locality improves as m increases. However, the real throughput is smaller than the optimal case and it depends on the scheduler. In Section 4, we use simulations to show that the optimal throughput is achievable. Thus, in analytical models in this paper, we denote the real throughput of m servers as $f(m) = m \cdot X(m)$.

2.3 Problem Statement

Our objective is to minimize the energy cost of the entire system, which consists of all MapReduce servers and fraction of web servers, shown in Equation (1). Two variables are m_t and m'_t, which need to fulfill MapReduce deadlines and web target response times. To capture the web target response times, we resort to simple $M/M/m'_t$ queuing model [5], i.e., find a minimum m'_t, such that response time $R_{w,t}$ under the arrival rate of $\lambda_{w,t}$ and the service rate of μ_w is below the target,

$$R_{w,t} = \frac{C(m'_t, \lambda_{w,t}/\mu_w)}{m'_t \mu_w - \lambda_{w,t}} + \frac{1}{\mu_w} \leq R^* , \qquad (3)$$

where C is the Erlang C Formula. Since we can derive the threshold for m'_t from analytical model, here we consider m'_t as a function of $\lambda_{w,t}$.

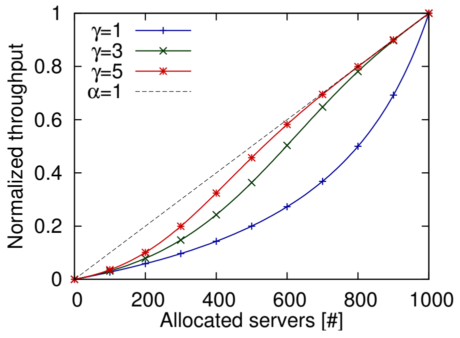


Figure 2: Normalized optimal throughput vs. number of allocated MapReduce servers: with different replication factors γ combined with remote slowdown $\alpha = 4$ and no remote slowdown $\alpha = 1$

As for interactive MapReduce, we require that MapReduce servers allocated in each control window are enough to serve incoming interactive MapReduce workloads, i.e., $f(m_t) \geq \tau \cdot \lambda_{c,t}$. And, for batch MapReduce tasks, they can be delayed by multiple control windows. Denote r_t as the aggregate residual batch tasks at the beginning of control window t, we have $r_1 = 0$, and

$$r_{t+1} = (r_t + \tau \cdot \lambda_{b,t} - (f(m_t) - \tau \cdot \lambda_{c,t})^+)^+, \;\; t = 2, \ldots, T \;, \qquad (4)$$

which means that interactive tasks $(\tau \cdot \lambda_{c,t})$ have higher priority to be served, and batch tasks $(r_t + \tau \cdot \lambda_{b,t})$ take the remaining capacity $(f(m_t))$.

In our problem, we consider 30 minutes as one control window and one day as the horizon of our problem. And we set the end of the day as the deadlines for all batch MapReduce tasks. The formal presentation of our problem is:

$$\begin{aligned} \min_{m_t} \quad & \sum_{t=1}^{T} K \cdot \max\{M, m_t + m'_t\} , \\ \text{s.t.} \quad & f(m_t) \geq \tau \cdot \lambda_{c,t}, \forall t , \\ & r_{T+1} = 0 , \;\; 0 \leq m_t \leq M, \forall t , \\ & \text{constraints } (3), (4) , \end{aligned} \qquad (5)$$

In each control window, the interactive MapReduce workloads determine the minimum number of m_t, while the flexibility of m_t comes from the delayable MapReduce tasks.

3. PERFECT WORKLOAD INFORMATION

We first solve the problem, assuming we have perfect knowledge of the future, namely, we know all the parameters $(\lambda_{b,t}, \lambda_{c,t}, \lambda_{w,t})$ at the beginning of time. Since the data locality improves as m increases, we assume $f(m)$ to be a *convex* and *strictly increasing* function.[1] Further, we assume the MapReduce cluster has enough servers to finish all the batch tasks within one day. Otherwise we should simply provision more servers to serve the workload, which is not the main focus of our work.

This optimization problem can be transformed to a concave minimization problem [10, 16], which is to minimize a concave objective function. In concave minimization problems, the local minimum lies on the boundary of the feasible region. Since there can

[1]Notice that with $\gamma \geq 3$, the optimal throughput function is non-convex. However, it can still be approximated by a convex function quite well, since its second order derivative keeps increasing in most part of the feasible regions when m is not close to M.

exist exponential number of local minima with the increase of the dimensions, traditional deterministic or randomized non-linear programming solvers cannot solve this kind of problems, even if we just want a sub-optimal numerical solution. Concave minimization problems belong to the "hard" global optimization problems. It has been proved that most concave minimization problems are NP-hard.

However, our problem has some special structures, such that we can solve it optimally using a greedy algorithm with complexity $O(T^2)$.

3.1 Problem Transformation

Since each server in the MapReduce cluster cannot be switched off at any time, the number of active servers cannot be smaller than M. Thus, the following lemma holds.

LEMMA 1. *In problem* (5)*, if* $(m_1^*,\dots,m_T^*)$ *is an optimal solution, then solution* $(\max\{m_1^*, M-m_1'\},\dots,\max\{m_T^*, M-m_T'\})$ *is feasible, and also optimal.* [2]

The problem can be transformed to the following problem.

$$\begin{aligned} \min_{m_t} \quad & \sum_{t=1}^{T} K\cdot(m_t+m_t')\ , \\ \text{s.t.} \quad & f(m_t) \ge \tau\cdot\lambda_{c,t},\ \forall t\ , \\ & r_{T+1}=0\ ,\ (M-m_t')^+ \le m_t \le M,\ \forall t\ , \\ & \text{constraints } (3),(4)\ . \end{aligned} \tag{6}$$

The following lemma follows directly from Lemma 1.

LEMMA 2. *Any optimal solution for Problem* (6) *is also an optimal solution for Problem* (5).

We define the inverse function of $f(m)$ as g, which is a *concave* and *strictly increasing* function. By replacing m_t, $f(m_t)$, $(M-m_t')^+$, M with $g(a_t)$, a_t, $g(l_t)$, $g(A)$, respectively, we rephrase Problem (6) as the following concave minimization problem:

$$\begin{aligned} \min_{a_t} \quad & \sum_{t=1}^{T} K\cdot(g(a_t)+m_t')\ , \\ \text{s.t.} \quad & a_t \ge \tau\cdot\lambda_{c,t},\ \forall t\ , \\ & r_{T+1}=0\ ,\ l_t \le a_t \le A,\ \forall t\ , \\ & \text{constraints } (3),(4)\ . \end{aligned} \tag{7}$$

3.2 Algorithm

We propose Algorithm 1 to solve the optimization problem, which is a two-stage greedy algorithm.

In the first stage, it greedily allocates the batch workload b_u into some later time t to achieve the throughput lower bound l_t. In the second stage, it goes backwards over time while greedily allocating the remaining batch workloads. The following theorem shows the complexity of this algorithm.

THEOREM 3. *If the feasible set is non-empty, the algorithm always finishes in* $O(T^2)$ *time.*

3.3 Optimality of the Algorithm

We denote $\mathscr{L}(b,c,l,A)$ as Problem (7) with parameters A, l_t, $\lambda_{b,t}=b_t/\tau$, and $\lambda_{c,t}=c_t/\tau$.[4]

[2]Details of the proofs can be found in Appendices.

[3]For argmin/argmax, we always break the tie arbitrarily.

[4]The m_t' terms in the objective function does not affect the optimal solution.

Algorithm 1 The Optimal Algorithm

```
1:  for $t$ from 1 to $T$ do
2:      $b_t \leftarrow \tau\cdot\lambda_{b,t}$
3:      $c_t \leftarrow \tau\cdot\lambda_{c,t}$
4:  end for
5:  for $v$ from 1 to $T$ do                                  ▷ Stage One
6:      if $c_v < l_v$ then
7:          for $u$ from $v$ to 1 do
8:              $\Delta \leftarrow \min\{b_u, l_v - c_v\}$
9:              if $\Delta > 0$ then
10:                 $b_u \leftarrow b_u - \Delta,\ c_v \leftarrow c_v + \Delta$
11:             end if
12:         end for
13:     end if
14:     $c_v \leftarrow l_v$
15: end for                                                 ▷ End of Stage One
16: for $u$ from $T$ to 1 do                                  ▷ Stage Two
17:     while $b_u > 0$ do
18:         $v \leftarrow \text{argmax}_t\{c_t \mid t \ge u, c_t < A\}$ [3]
19:         $\Delta \leftarrow \min\{b_u, A - c_v\}$
20:         $b_u \leftarrow b_u - \Delta,\ c_v \leftarrow c_v + \Delta$
21:     end while
22: end for                                                 ▷ End of Stage Two
23: for $t$ from 1 to $T$ do
24:     $m_t \leftarrow c_t$
25: end for
```

In each step of either stage, the algorithm reduces the current problem into another problem by decreasing b_u and increasing c_v. At the end of stage two, we get a problem such that $b_t = 0$ and $0 \le c_t \le A$ for each t, whose optimal solution is trivially $m_t = c_t$. By Theorem 3 and Theorem 4, it holds that the algorithm always returns the optimal solution.

THEOREM 4. *For any* $\mathscr{L}(b,c,l,A)$ *with non-empty feasible set, the algorithm keeps the optimal solution feasible in both stage one and stage two.*

For Theorem 4, it follows directly by Lemma 5, Lemma 6 and Lemma 7, which show that in Algorithm 1, Line 10, Line 14 and Line 24 keep the optimal solution feasible, respectively.

LEMMA 5. *For any problem* $\mathscr{L}(b,c,l,A)$*, if* $v=\min\{t \mid c_t < l_t\}$ *and* $u=\max\{t \mid t \le v, b_t > 0\}$ *exist, setting* c' *and* b' *by* $c_v' = c_v+\Delta$ *and* $b_u' = b_u - \Delta$ *for* $\Delta = \min\{b_u, l_v - c_v\}$*, while keeping all other elements in b and c unchanged, it holds that any optimal solution for* $\mathscr{L}(b',c',l,A)$ *is also an optimal solution for* $\mathscr{L}(b,c,l,A)$.

LEMMA 6. *For any problem* $\mathscr{L}(b,c,l,A)$*, if* $v=\min\{t \mid c_t < l_t\}$ *exists and* $b_t = 0$ *holds for any t, setting* c' *by* $c_t' = l_t$ *while keeping all other elements the same, the feasible sets of* $\mathscr{L}(b,c,l,A)$ *and* $\mathscr{L}(b,c',l,A)$ *are equivalent.*

LEMMA 7. *For any problem* $\mathscr{L}(b,c,l,A)$ *with a non-empty feasible set, if* $c_t \le l_t$ *holds for any t and* $u=\max\{u \mid b_u > 0\}$ *and* $v = \text{argmax}_t\{c_t \mid t \ge u, a_t < A\}$ *exist, setting* c' *and* b' *by* $c_v' = c_v + \Delta$ *and* $b_u' = b_u - \Delta$ *for* $\Delta = \min\{b_u, A - c_v\}$*, while keeping all other elements the same, it holds that any optimal solution for* $\mathscr{L}(b',c',l,A)$ *is also an optimal solution for* $\mathscr{L}(b,c,l,A)$.

Here we present the intuitions behind these lemmas. First, Lemma 5 tells us if at some time interactive MapReduce and web application workloads are not enough to make use of the whole MapReduce cluster, we should try to look for the batch tasks arriving no later

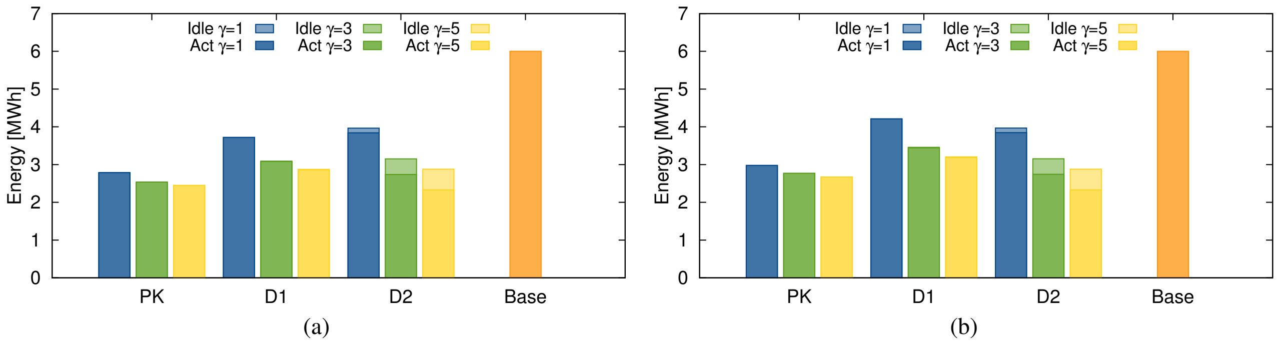


Figure 3: Energy consumption for the MapReduce cluster: (a) geometric arrivals and (b) day-night arrivals.

than that time to fill the residual cluster capacity, since MapReduce servers can not be turned off. Then, Lemma 7 tells us that for the remaining batch workloads we should greedily assign them at the control window which has already been assigned the largest MapReduce throughput, until achieving MapReduce cluster capacity. Since the cost function g is concave in the amount of admitted MapReduce workload a_t, the more we admit, the more energy-efficient we are.

4. EVALUATION

In this section, we use simulation to first verify our assumption on data locality. Second we compare our optimal perfect-knowledge (PK) controller with two dummy allocation schemes referred to as D1 and D2. In each control period, D1 allocates enough servers to complete the expected MapReduce workload, while D2 uses a constant allocation based on the average workload over the whole time horizon.

4.1 Experimental Setup

We consider a system comprising both a MapReduce cluster and a web farm of size 1000 and 1800, respectively, over a 1-day time horizon. At the beginning of each 30 minutes, the control window size τ, the controller decides how many servers m_t to allocate to both the batch and interactive MapReduce jobs, while the remaining MapReduce servers are used as web servers.

The web workload can not be controlled by the system and we assume to know the required web servers m'_t. Both batch and interactive MapReduce tasks are handled by a priority scheduler which dispatches them to the currently available servers. To satisfy the different delay requirements of batch and interactive MapReduce, the scheduler gives strict priority to interactive MapReduce.

The task service rate is homogeneous across all servers, but each server checks if the task being served is local or remote. For the latter the service rate is decreased by a slowdown factor $\alpha = 4$. Moreover, we set the unit energy cost per on/active server $K = 250$ Watt [18].

We consider two workload scenarios. The first scenario uses geometrical distributed arrival rates for batch MapReduce $\lambda_{b,t}$ and interactive MapReduce $\lambda_{c,t}$. The second scenario uses a day-night pattern where $\lambda_{b,t}$, $\lambda_{c,t}$ follow a sinusoidal pattern across the day. This pattern represents better the workloads monitored in real data-centers. In all cases inter-arrival rates and task execution times are exponentially distributed with mean $\frac{1}{\lambda}$ and $\mu = 10$s, respectively. Each task is randomly assigned to a data chunk which is uniformly distributed across the MapReduce servers with replication factor $\gamma = [1,3,5]$. Once a task is scheduled we use the probability $p_l(m)$ to decide if it is local or remote such as to achieve the optimal throughput function $f(m)$.

We repeat each experiment 50 times and present the mean values over all repetitions. The resulting confidence intervals are quite narrow – between +0.25% and -0.25% of the mean values in all cases except for batch MapReduce response times which have a maximum confidence interval between +7.8% and -7.8% of the mean.

4.2 Energy saving

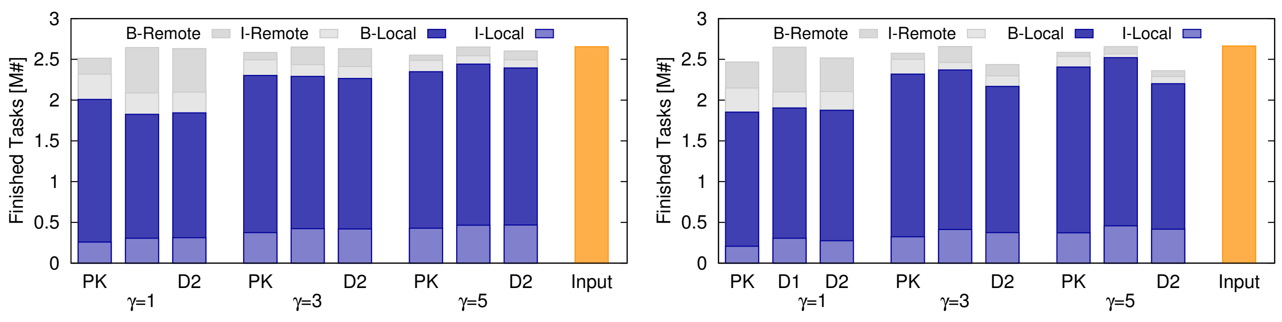
The objective of the controller is to minimize the system energy consumption through the number of allocated MapReduce servers m_t. We present the impact of the allocation decisions taken by each controller on the energy consumption in Figure 3 (a) and (b). As baseline we also present the always on energy consumption of the whole MapReduce cluster. Note that the consumed energy refers only to the MapReduce cluster since the number of web servers is not controlled and adds 23.9 MWh to all scenarios. The figures distinguish between power used for busy servers and power used for idle servers. Idle servers are possible because during low load periods the total workload of MapReduce and web does not suffice to saturate the capacity of the MapReduce cluster. One can observe that in both scenarios our PK scheduler outperforms the baseline by up to 59.3% and the dummy controllers by up to 30.7%. Moreover, one can observe higher energy savings when increasing the replication factor. A higher replication factor allows to raise the local probability and the achievable throughput and hence diminishes the number of servers required to satisfy the same demand.

4.3 Un-/Finished tasks/locality

While energy minimization is the optimization objective, we still want to satisfy all the requests coming to the system. Considering again all three controllers and the same scenarios as in our previous section, we depict in Figure 4 (a) and (b) the amount of finished tasks over the whole time horizon split per data locality and type and compare it to the input load. One can observe that the D1 controller is able to almost always complete all the tasks, while PK and, definitely, D2 lag behind. Here the PK controller suffers from its open-loop operation which prevents it to react to errors in each control period and these errors accumulate over time. Even if the D1 controller succeeds to finish all jobs, D1 lacks the energy saving possibilities given by the PK controller. Hence, PK achieves the trade-off between energy and finished jobs. In a practical scenario, the problem of unfinished tasks can be easily treated as additional workload in the first control period of the new time horizon.

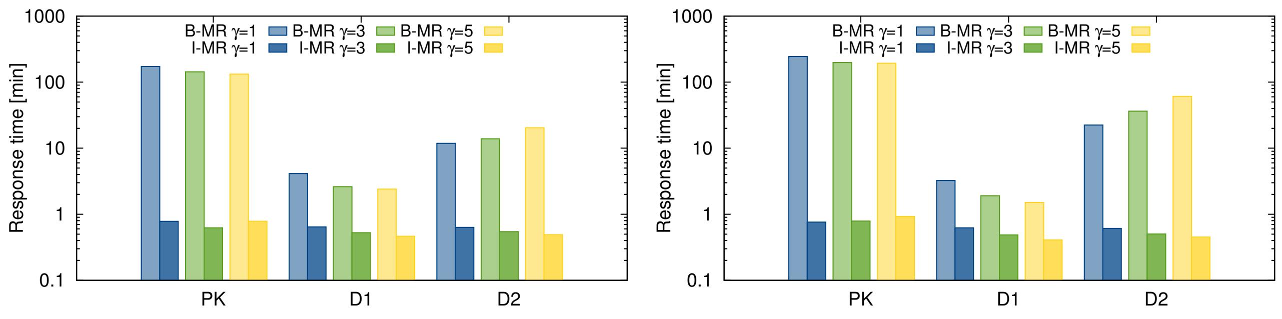
4.4 Response times

We conclude our evaluation presenting the effect of each control allocation on the mean response time. These are shown in Figure 5 (a) and (b) split by interactive and batch MapReduce. One can



(a) (b)

Figure 4: Finished tasks over 1-day vs. input load: (a) geometric arrivals and (b) day-night arrivals.



(a) (b)

Figure 5: Response time for (I)nteractive and (B)atch MapReduce (MR) tasks: (a) geometric arrivals and (b) day-night arrivals.

observe how the PK controller delays batch MapReduce by looking at their significant delay increment over the D1 and D2 cases. More in general, the best results are obtained by D1 followed by D2. Overall the energy minimizing allocation of the PK controller pays the price that, by reducing the number of MapReduce servers, it increases the MapReduce cluster load which negatively affects the response times. Even so, PK still manages to keep the interactive MapReduce delay increase limited.

5. RELATED WORK

There are plethora of studies aiming to improve energy efficiency for different types of systems, ranging from conventional web service systems [7, 13] to modern big data systems [6, 11, 12, 15, 19], such as MapReduce. To minimize energy consumptions, server resources are dynamically provisioned and workloads are scheduled correspondingly. Thereby, we summarize the related work in the area of dynamic sizing with special focus on MapReduce systems.

Dynamic sizing is proven effective for workloads that show strong time varying patterns, e.g., switching on/off servers for web services [7, 13]. Due to the issues of data accessibility of underlying file systems [2] and performance dependency on data locality [9], dynamic sizing is applied on in a partial manner. Current work either control the fraction of time of the entire cluster [11] or the fraction of servers all the time [12]. On the one hand, to harvest the maximum data locality, data intensive workloads are batched and executed together on the MapReduce clusters [6, 11] for certain duration, often at midnight. On the other hand, another set of studies [6, 12] leverage the data replication factor (e.g., 3 replicas per data [3]) and propose the concept of covering set, which keeps only a fraction of servers on all the time. Certain modifications on file systems are usually needed.

To address the emerging class of interactive MapReduce workloads, BEEMR [6] combines the merits of both types of approaches by partitioning the MapReduce clusters into two zones, namely interactive and batch. However, how to optimally (and dynamically) allocate and execute interactive and batch MapReduce is yet to be discussed. Motivated by the complimentary performance requirements of service and data-intensive workloads, another hosts of studies [14, 17, 22] try to schedule MapReduce workloads according to the dynamics of web workloads, considering only the batch MapReduce and (often) overlooking the locality issues. All in all, the related work falls short in addressing how to dynamically size the server resources for executing web, interactive and batch MapReduce in an energy optimal fashion.

Overall, it is not clear how to design a scheduler that can consider multiple types of workloads with different performance and system requirements, i.e., delay tolerances and data locality, in conjunction with dynamic sizing.

6. CONCLUSION AND SUMMARY

This study considers both control knobs of dynamic sizing and scheduling policies simultaneously, for three types of workloads: web service, batch MapReduce and interactive MapReduce. We are able to achieve minimum energy consumption by allocating optimal servers across the three types of workloads, factoring their data locality, delay constraints, and workload uncertainties, while dynamically scheduling/delaying batch MapReduce. We developed an optimal algorithm under perfect knowledge with complexity $O(T^2)$. By means of extensive simulation, we show energy savings of up to 30% and 59% over dumb and no allocation strategies, respectively, without affecting interactive MapReduce delay significantly.

7. REFERENCES

[1] Hadoop. `http://hadoop.apache.org`. Accessed: 2010-09-30.

[2] Hadoop HDFS. `http://hadoop.apache.org/docs/r1.2.1/hdfs\_design.html`. Accessed: 2010-09-30.

[3] How Google Works. `http://www.baselinemag.com/c/a/Infrastructure/How-Google-Works-1/`. Accessed: 2010-09-30.

[4] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB: Queries with Bounded Errors and Bounded Response Times on Very Large Data. In *ACM EuroSys*, pages 29–42, 2013.

[5] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley-Interscience, New York, NY, USA, 1998.

[6] Y. Chen, S. Alspaugh, D. Borthakur, and R. H. Katz. Energy efficiency for large-scale MapReduce workloads with significant interactive analysis. In *EuroSys*, pages 43–56, 2012.

[7] Y. Chen, A. Das, W. Qin, A. Sivasubramaniam, Q. Wang, and N. Gautam. Managing server energy and operational costs in hosting centers. In *SIGMETRICS*, pages 303–314, 2005.

[8] B. Cho, M. Rahman, T. Chajed, I. Gupta, C. Abad, N. Roberts, and P. Lin. Natjam: Design and Evaluation of Eviction Policies for Supporting Priorities and Deadlines in Mapreduce Clusters. In *ACM SoCC*, pages 6:1–6:17, 2013.

[9] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears. MapReduce Online. In *NSDI*, pages 313–328, 2010.

[10] R. Horst. On the global minimization of concave functions. *Operations Research Spektrum*, 6(4):195–205, 1984.

[11] W. Lang and J. M. Patel. Energy Management for MapReduce Clusters. *PVLDB*, 3(1):129–139, 2010.

[12] J. Leverich and C. Kozyrakis. On the energy (in)efficiency of Hadoop clusters. *Operating Systems Review*, 44(1):61–65, 2010.

[13] M. Lin, A. Wierman, L. L. H. Andrew, and E. Thereska. Dynamic right-sizing for power-proportional data centers. In *INFOCOM*, pages 1098–1106, 2011.

[14] Z. Liu, Y. Chen, C. Bash, A. Wierman, D. Gmach, Z. Wang, M. Marwah, and C. Hyser. Renewable and cooling aware workload management for sustainable data centers. In *SIGMETRICS*, pages 175–186, 2012.

[15] N. Maheshwari, R. Nanduri, and V. Varma. Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework. *Future Generation Comp. Syst.*, 28(1):119–127, 2012.

[16] P. M. Pardalos and J. B. Rosen. Methods for global concave minimization: A bibliographic survey. *Siam Review*, 28(3):367–379, 1986.

[17] B. Sharma, T. Wood, and C. R. Das. HybridMR: A Hierarchical MapReduce Scheduler for Hybrid Data Centers. In *ICDCS*, pages 102–111, 2013.

[18] D. Wang, S. Govindan, A. Sivasubramaniam, A. Kansal, J. Liu, and B. Khessib. Underprovisioning backup power infrastructure for datacenters. In *Proceedings of the 19th international conference on Architectural support for programming languages and operating systems*, pages 177–192. ACM, 2014.

[19] T. Wirtz and R. Ge. Improving mapreduce energy efficiency for computation intensive workloads. In *IGCC*, pages 1–8, 2011.

[20] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-tolerant Abstraction for In-memory Cluster Computing. In *USENIX NSDI*, pages 2–2, 2012.

[21] W. Zhang, S. Rajasekaran, and T. Wood. Big data in the background: Maximizing productivity while minimizing virtual machine interference. In *ASDB*, 2013.

[22] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu. MIMP: Deadline and Interference Aware Scheduling of Hadoop Virtual Machines. In *CCGRID*, pages 394–403, 2014.

APPENDIX

A. PROOF FOR LEMMA 1

PROOF. Since $(m_1^*,\dots,m_T^*)$ is an optimal solution, it meets all the constraints and achieves the minimal value of the objective function. Consider the solution $(\max\{m_1^*,M-m_1'\},\dots,\max\{m_T^*,M-m_T'\})$. At each control window, $\max\{m_t^*,M-m_t'\}$ will be no smaller than m_t^*, which means it can also finish all the tasks at the end of the control horizon. Thus, it is also a feasible solution. Moreover, since this solution has exactly the same value of the objective function as the original one, it is also an optimal solution. $\square$

B. PROOF FOR THEOREM 3

PROOF. The runtime of stage one is trivially $O(T^2)$. It suffices to show that stage two finishes in $O(T^2)$ time. Or more precisely, each time the WHILE loop (Line 17-20) finishes in linear time. In each iteration of the WHILE loop, we have either $\Delta = b_u$ or $\Delta = A - c_v$. The algorithm either finishes the WHILE loop or removes c_v from the sorted list. Using the trick of sorting the $\{c_1,\dots,c_T\}$ before stage two, if in stage two the set $\{c_t | t \geq u, c_t < A\}$ is never empty, then the WHILE loop terminates in linear time.

If at some time during the stage two, the set $\{c_t | t \geq u, c_t < A\}$ is empty, then the problem with the b,c,l,A values at that time has empty feasible set. Thus, it suffices to show that if the original problem has a non-empty feasible set, each step in stage one or stage two keeps the feasible set non-empty. This holds by Lemma 8, 9 and 6. $\square$

LEMMA 8. *In Algorithm 1, Line 10 keeps the feasible set of $\mathscr{L}(b,c,l,A)$ non-empty. Namely, if before this operation, $\mathscr{L}(b,c,l,A)$ has a non-empty feasible set, then after this operation its feasible set is still non-empty.*

PROOF. The feasible set of problem $\mathscr{L}(b,c,l,A)$ is non-empty if and only if for any k

$$c_k < A \ , \quad \sum_{t=k}^{T}(b_t + c_t) \leq (T-k+1)\cdot A \ . \tag{8}$$

It suffices to show Line 10 keeps these properties. First, since Line 10 ensures $c_v \leq l_v = f(M - m_t') \leq A$, the first property $c_v < A$ is kept. Then, we show that Line 10 also keeps the second property. One can verify that in Line 10, if $u < v$, then $b_{u+1},\dots,b_v = 0$, so that $b_t + c_t \leq A$ holds for $t \in \{u+1,\dots,v\}$, which shows the second property is kept; otherwise $u = v$, then $b_v + c_v$ is unchanged. $\square$

LEMMA 9. *In Algorithm 1, Line 20 keeps the feasible set of $\mathscr{L}(b,c,l,A)$ non-empty. Namely, if before this operation, $\mathscr{L}(b,c,l,A)$*

has a non-empty feasible set, then after this operation its feasible set is still non-empty.

PROOF. The proof is similar. It suffices to show Line 20 keeps (8). Since in Line 20, $b_{u+1},\dots,b_T = 0$, it holds that $b_t + c_t = c_t \le A$ for $t \in \{u+1,\dots,T\}$. Thus, if we increase c_v for some $v > u$, (8) still holds. Otherwise $u = v$, then $b_v + c_v$ is unchanged. $\square$

C. PROOFS FOR LEMMAS 5, 6 AND 7

Before showing formal proofs, we need the following lemmas.

LEMMA 10. *For the function $f(x,y) = (x-y)^+$, for any y and $\Delta \ge 0$, it holds that*

$$x - x' \le \Delta \implies f(x,y) - f(x',y) \le \Delta\,.$$

LEMMA 11. *For the function $f(x,y) = (x-y)^+$, for any x and $\Delta \ge 0$, it holds that*

$$y - y' \le \Delta \implies f(x,y') - f(x,y) \le \Delta\,.$$

We omit the proofs for Lemma 10 and 11, since they hold trivially.

LEMMA 12. *For any problem $\mathscr{L}(b,c,l,A)$, if we set c' and b' the same as c and b except that $c'_v = c_v + \Delta$ and $b'_u = b_u - \Delta$, for some u,v such that $u \le v$ and some non-negative Δ, and still keeping $c'_v \le A$, $b'_u \ge 0$, it holds that the feasible region of $\mathscr{L}(b',c',l,A)$ is a subset of the feasible region of $\mathscr{L}(b,c,l,A)$*

PROOF. Consider an arbitrary feasible solution of $\mathscr{L}(b',c',l,A)$, $a' = (a'_1,\dots,a'_T)$. Denote that $r_1 = 0$, $r_{t+1} = (r_t + b_t + c_t - a'_t)^+$ for problem $\mathscr{L}(b,c,l,A)$, and $r'_1 = 0$, $r'_{t+1} = (r'_t + b'_t + c'_t - a'_t)^+$ for problem $\mathscr{L}(b',c',l,A)$. If $u = v$, the lemma holds trivially since $r'_t = r_t$ for any t. So now we consider the case that $u < v$. Notice that for any $t \le u$, $r'_t = r_t$. Since $b'_u = b_u - \Delta$, by Lemma 10, we have $r'_{u+1} \ge r_{u+1} - \Delta$. Then by induction we have $r'_t \ge r_t - \Delta$ for any $t \in \{u+1,\dots,v\}$. Since $c'_v = c_v + \Delta$, we have $r'_{v+1} \ge r_{v+1}$. By induction we have $r'_{T+1} \ge r_{T+1}$, which means $r'_{T+1} = 0$ implies $r_{T+1} = 0$. Thus, if a' is a feasible solution of $\mathscr{L}(b',c',l,A)$, then it is also a feasible solution of $\mathscr{L}(b,c,l,A)$. $\square$

Now we present the formal proof for Lemma 5, 6 and 7.

C.1 Proof for Lemma 5

Lemma 5 follows directly by the following lemma.

LEMMA 13. *For problem $\mathscr{L}(b,c,l,A)$, assume there exists c_v which has the smallest index v such that $c_v < l_v$, and there exists b_u which has the largest index u such that $u \le v$ and $b_u > 0$. Denote that $\Delta = \min\{b_u, l_v - c_v\}$. Set c' and b' the same as c and b, except that $c'_v = c_v + \Delta$ and $b'_u = b_u - \Delta$. It holds that any optimal solution for $\mathscr{L}(b',c',l,A)$ is also an optimal solution for the original problem.*

PROOF. For notation simplicity, in this proof we use $\mathscr{L}$ to denote $\mathscr{L}(b,c,l,A)$, and use $\mathscr{L}'$ to denote $\mathscr{L}(b',c',l,A)$. It follows by Lemma 12 that the feasible region of $\mathscr{L}$ is a subset of the feasible region of $\mathscr{L}'$. Moreover, these two problems have the same objective function. Thus, it suffices to show any optimal solution of $\mathscr{L}$, $a^* = (a^*_1,\dots,a^*_T)$, is also a feasible solution of $\mathscr{L}'$. We show this in two cases: $u = v$ and $u < v$. In the former case, the two problems are equivalent since $c_v < c'_v \le l_v$ and $c_v + b_v = c'_v + b'_v$. For the latter case that $u < v$, assume $(r_1,\dots,r_{T+1})$ and $(r'_1,\dots,r'_{T+1})$ are values for residual batch tasks corresponding to the solution a^* in $\mathscr{L}$ and $\mathscr{L}'$, respectively. With the same argument in the proof of Lemma 12, we get that

$$\begin{cases} r'_t = r_t\,, & t \le u\,,\\ r'_t = (r_t - \Delta)^+\,, & u < t \le v\,. \end{cases} \tag{9}$$

Now we show the following claim:

CLAIM 1. *If a^* is an optimal solution for $\mathscr{L}$, it holds that $r_t \ge \Delta$ for $u < t \le v$.*

We argue this by contradiction. Assume the statement does not hold, then there must exist some k, such that $u \le k < v$ and $r_{k+1} = (r_k + b_k + c_k - a^*_k)^+ < \Delta$. Without loss of generality, we assume k is the smallest one of all the possible values and assume $r_{k+1} = \Delta - \delta$. Then, it holds that

1. $a^*_k \ge c_k + \delta$,
2. $r_v + b_v + c_v - a^*_v \le -\delta$.

The first statement holds because $r_k + b_k \ge \Delta$ (if $k = u$, it is also true because $b_u \ge \Delta$), and $(r_k + b_k + c_k - a^*_k)^+ = \Delta - \delta$. The second statement holds because $r_v \le \Delta - \delta$, $b_v = 0$ and $a^*_v \ge c_v + \Delta$, where $r_v \le \Delta - \delta$ holds because $r_{k+1} \ge \dots \ge r_v$, following by the fact that $b_{k+1} = \dots = b_{v-1} = 0$ and $a_t \ge c_t$ for any t, and $a^*_v \ge c_v + \Delta$ holds because $a^*_v \ge l_v$ and $\Delta = \min\{b_u, l_v - c_v\}$. Then we get a better solution a^{**} for $\mathscr{L}$, such that

$$a^{**}_t = \begin{cases} a^*_t - \delta, & t = k\\ a^*_t, & \text{otherwise} \end{cases}.$$

This solution still meets all the constraints of $\mathscr{L}$, because 1) $a^*_k - \delta \ge c_k \ge l_k$, and 2) $r^{**}_{v+1} = r_{v+1},\dots,r^{**}_T = r_T$, where r^{**}_t are residual batch tasks corresponding to a^{**}. To show the latter statement, notice that by Lemma 11, $a^{**}_t = a^*_t - \delta$ implies $r^{**}_{k+1} \le r_{k+1} + \delta$; by Lemma 10, this further implies $r^{**}_v \le r_v + \delta$. Thus, we have $r^{**}_v + b_v + c_v - a^{**}_v \le (r_v + \delta) + b_v + c_v - a^*_v \le 0$, which implies $r^{**}_{v+1} = r_{v+1} = 0$. And $r^{**}_{v+2} = r_{v+2},\dots,r^{**}_T = r_T$ follows since $a^{**}_t = a^*_t$ for $t = v+1,\dots,T$. We have constructed the contradiction. Thus, Claim 1 holds.

Combining Claim 1 with (9), we get that $r'_v = r_v - \Delta$, which implies $r'_{v+1} = r_{v+1}$ (recall $c'_v = c_v + \Delta$). Since from time slot $v+1$ the two problems $\mathscr{L}$ and $\mathscr{L}'$ are exactly the same, we have $r'_{T+1} = r_{T+1} = 0$, which means a^* is also a feasible solution of $\mathscr{L}'$. $\square$

C.2 Proof for Lemma 6

Lemma 6 follows directly by the following lemma.

LEMMA 14. *For problem $\mathscr{L}(b,c,l,A)$, if there exists c_v, $v \in \{1,\dots,T\}$, such that $c_v < l_v$, and for any $t \le v$ we have $b_t = 0$, then by setting c' same as c except that $c'_v = l_v$, we get a new problem $\mathscr{L}(b,c',l,A)$, whose feasible set is equivalent to the feasible set of $\mathscr{L}(b,c,l,A)$.*

PROOF. We can clearly see that for any feasible solution a for $\mathscr{L}(b,c,l,A)$, since $a_t \le \max\{c_t, l_t\}$ and $b_t = 0$ for $t \in \{1,\dots,v\}$, we have that $r_{v+1} = r'_{v+1} = 0$. By induction, it holds that $r_t = r'_t = 0$ for any $t > v$. Thus a is also a feasible solution for $\mathscr{L}(b,c',l,A)$. $\square$

C.3 Proof for Lemma 7

We need the following “non-slackness” lemma and ”concave minimum” lemma, which are the two key insights of Lemma 7.

LEMMA 15 (NON-SLACKNESS). *For any problem $\mathscr{L}(b,c,l,A)$ such that $c_t \ge l_t$ for any t, the optimal solution $a^* = (a^*_1,\dots,a^*_T)$ will make $r_t + b_t + c_t - a^*_t \ge 0$ for any t.*

PROOF. The lemma holds, since otherwise we can decrease a^*_t. $\square$

LEMMA 16 (CONCAVE-MINIMUM). *For a concave function $g(x)$, given any x_1,x_2,x_3,x_4, such that $x_1 \le x_3 \le x_4$ and $x_1 + x_4 = x_2 + x_3$, it holds that $g(x_1) + g(x_3) \le g(x_2) + g(x_3)$.*

PROOF. This lemma follows directly by the definition of the concave function. □

Lemma 7 follows directly by the following lemma.

LEMMA 17. *For any problem $\mathscr{L}(b,c,l,A)$ such that $c_t \geq l_t$ for any t, assume b_u is the largest index u such that $b_u > 0$, and v is the index such that $v = argmax_t\{c_t |\ t \geq u, a_t < A\}$ (break the tie arbitrarily). Denote that $\Delta = \min\{b_u, A - c_v\}$. Set c' and b' same as c and b, except that $c'_v = c_v + \Delta$ and $b'_u = b_u - \Delta$. It holds that any optimal solution for $\mathscr{L}(b',c',l,A)$ is also an optimal solution for $\mathscr{L}(b,c,l,A)$.*

PROOF. For notation simplicity, we use $\mathscr{L}$ to denote $\mathscr{L}(b,c,l,A)$, and use $\mathscr{L}'$ to denote $\mathscr{L}(b',c',l,A)$.

First we show that an optimal solution of $\mathscr{L}$ exists such that $a_v \geq c_v + \Delta$. Assume we have an optimal solution a^* for $\mathscr{L}$, such that $a^*_v < c_v + \Delta$, then there must exist $v' \geq u$ ($v' \neq v$) such that $a^*_{v'} > c_{v'}$, otherwise we cannot finish all batch tasks at the end. We set $a^{**}_v = a^*_v + \delta$ and $a^{**}_{v'} = a^*_{v'} - \delta$, where $\delta = \min\{a^*_{v'} - c_{v'}, A - a^*_v\}$, and keep $a^{**}_t = a^*_t$ for other t's. Then, it holds that a^{**} is also a feasible solution since it meets the constraints and can also complete all tasks at the end[5].

Observe that $\delta = a^*_{v'} - c_{v'}$ implies $c_{v'} = a^{**}_{v'} \leq a^*_v \leq a^{**}_v$, since by definition $c_{v'} \leq c_v$; while $\delta = A - a^*_v$ implies $a^{**}_{v'} \leq a^*_{v'} \leq a^{**}_v = A$. Either case implies $g(a^{**}_{v'}) + g(a^{**}_v) \leq g(a^*_{v'}) + g(a^*_v)$ (see Lemma 16), which shows a^{**} is at least as good as a^*. We can do this kind of adjustment again and again, without decreasing the value of the objective function, until either $a_v = A$ (saturated), or $a_t = c_t$ for any $t \in \{u, \ldots, T\}/\{v\}$, which indicates that we can eventually get an optimal solution such that $a_v \geq c_v + \Delta$.

Then we show that if an solution of $\mathscr{L}$ satisfying that $a_v \geq c_v + \Delta$, then it is also a feasible solution of $\mathscr{L}'$, because 1) the constraints $c'_t \leq a_t \leq A$ still hold (we only need to check the case when $t = v$); 2) in order to show $r'_{T+1} = 0$, we have

$$\begin{aligned}
r'_{T+1} &= \sum_{t=u}^{T} (r'_t + b'_t + c'_t - a_t)^+ \\
&\overset{(a)}{=} ((r'_u + b'_u) + \sum_{t=u}^{T} c'_t - \sum_{t=u}^{T} a_t)^+ \\
&\overset{(b)}{=} ((r_u + b_u - \Delta) + \Delta + \sum_{t=u}^{T} c_t - \sum_{t=u}^{T} a_t)^+ \\
&= r_{T+1} = 0\,,
\end{aligned}$$

where (a) holds since $b_{u+1} = \ldots = b_T = 0$, and (b) holds since $b'_u = b_u - \Delta$, $c'_v = c_v + \Delta$ and $r'_u = r_u$ (we can use induction to show $r'_t = r_t$ for all $t \leq u$). □

[5]Recall that by definition we have $b_t = 0$ for any $t \in \{u+1, \ldots, N-1\}$, so as long as the constraints $c_t \leq a_t \leq A$ are met and we also have $r_u + b_u = (a_u - c_t) + \ldots + (a_T - c_T)$, it can finish all the tasks at the end so it is a feasible solution.