Optimizing Energy, Locality and Priority in a MapReduce Cluster

Yijun Ying*, Robert Birke†, Cheng Wang‡, Lydia Y. Chen† and Gautam Natarajan§
*EPFL, †IBM Research Zurich, ‡Penn State University, §Texas A&M University

***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. In particular, we resort to techniques of Markovian Decision Process (MDP) and devise a throughput optimal scheduler supporting the scheme. Finally, using extensive simulation results, we show that the proposed methodology can efficiently minimize the datacenter energy cost while fulfilling the delay constraints of workloads.**

I. 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 [2], [3], MapReduce workloads evolve from traditional throughput sensitive batch jobs to increasingly delay sensitive interactive jobs, such as Sparks [4] on stream processing, Natjam [5] on batch, and BlinkDB [6] 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 [7] 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 [8], [9]. 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 [10] 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 [12]), a few solutions modify the underlying file system so that a fraction of servers, e.g., one third of the servers [13], 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 [3], 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 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. We tackle this problem by considering two scenarios, i.e., perfect workload and uncertain workload information. 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. To deal with workload uncertainty in real systems, we resort to finite Markovian Decision Process (MDP) for modeling a certain length of optimization horizon, e.g., one day. We show the special properties of the optimal policy which can be used for analyzing the tradeoffs between the optimal solutions and workload characteristics. We devise a data-locality aware scheduler to support our system which overcomes two limitations of [14], i.e., throughput stability in the overload region and task prioritization support. Finally, we build an event driven simulator to empirically prove the throughput optimality of the scheduler and to evaluate the proposed algorithms 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 II provides an overview of the system and a formal definition of the problem statement. The algorithm for perfect workload information is detailed in Section III. We present MDP solutions in Section IV. Section V details the new data-locality-aware scheduler and its empirical proof of throughput optimality. Simulations results are discussed in Section VI. Section VII compares related work, followed by summaries and conclusions in Section VIII.

II. Workload, System Architecture and Problem Statement

In this section, we describe the workload characteristics, system architecture and assumptions considered and formally present the problem statement.

A. Workload and 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. Their proprieties are determined by their sensitivities to the latency and the resulting priority order is web, interactive MR, followed by batch MR. 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.

The high level schematic of the system architecture is depicted in Figure 1. We consider two types of management modules, namely controller and scheduler, which manages the number of nodes for different workloads and the scheduling decision, respectively. When different tasks from different workloads 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. We design a locality aware scheduling policy, which can specifically deal with transient overloads when the number of available of servers is low.

To control the server allocation for three workloads, 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.

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 [15], [16], 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

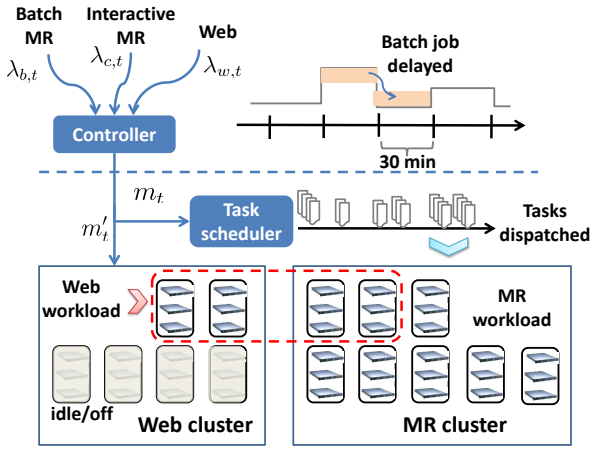

Fig. 1. High level system schematic. The controller takes as input the workload characteristics and outputs dynamic server allocation decisions for the three types of workloads; the scheduler then dispatches tasks according to different scheduling policies onto the lower level.

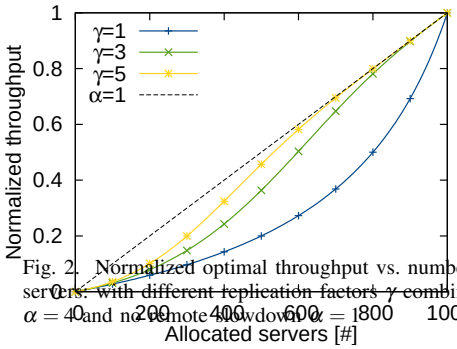

Fig. 2. Normalized optimal throughput vs. number of allocated MapReduce servers, with different replication factors γ combined with remote slowdown $\alpha = 4$ and no remote slowdown at $\alpha = 1$.

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 is the number of time windows, and K is the unit energy cost per *on/active* server. Note that we assume the energy cost of *off/inactive* servers to be zero.

B. The 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, $1 \le \gamma \le M$. 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 $\gamma > M - m$, 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. We devise a scheduler satisfying this assumption in Section V.

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

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

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 $m \cdot X(m)$, which can be achieved by locality aware scheduler [26]. 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. When developing the controller of server allocation, we assume that there exist the locality-aware schedule, which can sustain the optimal throughput of m servers, i.e., $f(m) = m \cdot X(m)$.

C. Problem Statement

Our objectives here are two folds: (1) how to minimize the energy cost of the entire system, under the assumption of achieving the optimal locality/throughput, and (2) how to design a scheduling policy that achieves the optimal locality and throughput shown in 2. Here, we focus on deriving the formal presentation of the first problem. The system of interest consists of all MapReduce servers and fraction of web servers. Two control variables are m_t and m'_t, that aim to 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 [17], 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,

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

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 $\lambda_{w,t}$.

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

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

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 specifically consider $\tau = 30$ minutes as one control window and one day as the horizon of our problem, i.e., $T = 48$. And we set the end of the day as the deadlines for all batch MapReduce tasks. The formal presentation of our problem is:

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

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.

III. Controller

In this section, we describe two controlling algorithms that determine the allocation of servers to web and MapReduce workloads, and the execution timing of batch MapReduce. The first algorithm is kind of off-line approach, based on perfect workload information in the future. To deal with workload uncertainty in the real life, we develop a MDP-based algorithm, via Markove Decision Processes. We term them PK and MDP algorithms in the remainder of the paper.

A. PK Algorithm

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.

[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.

Algorithm 1 The Optimal Off-line Algorithm

```
1: for t from 1 to T do
2:     b_t ← τ · λ_{b,t}
3:     c_t ← τ · λ_{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:             Δ ← min{b_u, l_v − c_v}
9:             if Δ > 0 then
10:                b_u ← b_u − Δ,  c_v ← c_v + Δ
11:            end if
12:        end for
13:    end if
14:    c_v ← 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 ← argmax_t{c_t | t ≥ u, c_t < A}  [2]
19:        Δ ← min{b_u, A − c_v}
20:        b_u ← b_u − Δ,  c_v ← c_v + Δ
21:    end while
22: end for                                               ▷ End of Stage Two
23: for t from 1 to T do
24:    m_t ← c_t
25: end for
```

This optimization problem can be transformed to a concave minimization problem [18], [19], 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 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. Fortunately, our problem has some special structures, such that we can solve it optimally using a greedy algorithm, summarized in Algorithm 1, with complexity $O(T^2)$. The detailed proof of Theorem 4 can be found in [20].

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 intuitions behind the algorithm. First, line 10 in algorithm 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 than that time to fill the residual cluster capacity, since MapReduce servers can not be turned off. Then, line 24 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

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

is concave in the amount of admitted MapReduce workload a_t, the more we admit, the more energy-efficient we are.

B. MDP Algorithm

The optimal algorithm in the previous subsection requires perfect knowledge about the future, since it solves the problem backwards over time. However, in the real world, workload arrivals can often be well modeled as stochastic processes, often Markovian. Therefore, we apply Markov Decision Process (MDP) approach to optimize the expected energy costs over finite horizon by dynamic sizing while leaving low-level fine-grained optimization, e.g., task priority, to the scheduler (designed in Section).

1) MDP Formulation: We consider a finite control horizon, e.g., one day, to capture the repetitive patterns of the workloads. We assume that batch MapReduce tasks have to be finished by the end of the control horizon. Recall that we have three types of workload: web services, interactive MapReduce and batch MapReduce, with task arrival rates $\lambda_{w,t}$, $\lambda_{b,t}$ and $\lambda_{c,t}$, respectively. Here, we assume that the time series of $\lambda_{w,t}$, $\lambda_{b,t}$ and $\lambda_{c,t}$ can be modeled as either a Markov process or a regular pattern (e.g., diurnal pattern) plus some Markovian noises. We further assume that at time t, we can accurately predict the arrival rates of this control window, but not the future.

Without perfect knowledge, we may have unfinished batch tasks at the end of the last control window, which does not satisfy the constraint $r_{T+1} = 0$ in Problem (5). To make the problem solvable, we remove this hard constraint and add a terminal cost term $v(r_{T+1})$ large enough for unfinished batch tasks as penalty. Intuitively, delaying tasks to the end of the control horizon should always have a marginal cost larger than admitting it earlier. In evaluation (see Section VI), we set a large marginal cost such that the controller will allocate enough servers to finish almost all the tasks at the end. Nevertheless, the quality of our solution is thus limited by such an assumption, i.e., we tend to process the batch MR more aggressively toward the end of the control horizon and miss the opportunity to further delay their execution.

Formally, we have:

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

2) States, Transition Probabilities and optimality equations: We define the system state variable at time t as 4-tuple $s_t = (\lambda_{w,t}, \lambda_{c,t}, \lambda_{b,t}, r_t)$. One can compute the transition probabilities from the system equation

$$s_{t+1} = (\lambda_{w,t+1}, \lambda_{c,t+1}, \lambda_{b,t+1}, \underbrace{(r_t + \tau\lambda_{b,t} - (f(m_t) - \tau\lambda_{c,t})^+)^+}_{r_{t+1}}) \ ,$$

assuming the conditional probabilities of $\mathsf{Pr}(\lambda_{i,t+1}|\lambda_{i,t})$, $i \in \{w, b, c\}$, are known.

Denote that the optimal cost-to-go function as $u_t^*(s_t)$. The optimality equation is

$$u_t^*(s_t) = \min_{m_t} \left\{ \mathop{\mathbb{E}}_{s_{t+1}} \left(K \cdot \max\{m_t + m_t', M\} + u_{t+1}^*(s_{t+1}) | s_t, m_t \right) \right\} , \tag{6}$$

where m_t' is derived from $\lambda_{w,t}$, with boundary condition

$$u_T^*(s_T) = \min_{m_T} \left\{ K \cdot \max\{m_T + m_T', M\} + v(r_{T+1}) \right\} \ . \tag{7}$$

The backward induction algorithm solves this problem in $O(L_m \cdot L_\lambda^4 \cdot T^2)$, where L_m, L_λ are the numbers of discretization levels for m_t and λ's, respectively.

If we further assume that the arrival rates ($\lambda_{w,t}$, $\lambda_{b,t}$, $\lambda_{c,t}$) are *stage independent*, we can define the system state more compactly, as the 2-tuple

$$s_t = (s_{1,t}, s_{2,t}) = (r_t + \tau\lambda_{b,t} + \tau\lambda_{c,t} \ , \ \max\{\tau\lambda_{c,t}, f(M - m_t')\}).^3$$

The first element represents the total amount of MapReduce workloads at time t. The second element represents the lower bound of the allocated MapReduce throughput at time t, which affects the action space (feasible set of m_t). The system equation is

$$\begin{cases} s_{1,t+1} = (s_{1,t} - f(m_t))^+ + \tau\lambda_{b,t+1} + \tau\lambda_{c,t+1} \\ s_{2,t+1} = \max\{\tau\lambda_{c,t+1}, f(M - m_{t+1}')\} \end{cases} ,$$

where m_t is the action, and the randomness comes from $\lambda_{m,t+1}$ $\lambda_{b,t+1}$ and $\lambda_{c,t+1}$. The optimality equation is the same as (8) and (9). In each iteration of the backward induction, before enumerating all possible values of m_t, we can reduce the size of the transition matrix by first computing the joint distribution of $\lambda_{b,t+1} + \lambda_{c,t+1}$. Thus, with the assumptions of stage-independent arrival rates, the MDP offers a runtime of $O(L_m \cdot L_\lambda^2 \cdot T^2)$.

3) Optimal Structure: Because of the non-linear cost function, the optimal solution of this problem does not have a simple structure. It is not guaranteed that the optimal policy is monotone in $\lambda_{b,t}$, $\lambda_{c,t}$ or r_t, even assuming the stage independence. However, we show the monotony of the optimal policy with independently exponentially distributed $\lambda_{b,t}$ and $\lambda_{c,t}$[4].

Theorem 1: Assuming $\lambda_{b,t}$ and $\lambda_{c,t}$ are stage independent and are sampled from exponential distributions, there exists an optimal policy increasing in $\lambda_{b,t}$, $\lambda_{c,t}$ and r_t.

Indeed, since the data locality improves as m_t increases, at some point, the optimal action m_t^* begins to increase very steeply as the arrival rates increase, until reaching M. The optimal policy is near-threshold based.

We also show that the optimal policy is decreasing in the arrival rate of web service tasks in any general case:

Theorem 2: An optimal policy decreasing in $\lambda_{w,t}$ exists.

Remark. Numerical examples show that the optimal policy is near threshold-based with respect to residual batch MapReduce workloads, arrival rates of the interactive/batch MapReduce workloads and web workloads. If there are more residual

[3] To simplify the notation, here we define that $f(m) = 0$ when $m < 0$.

[4] Details of the proofs can be found in Appendix.

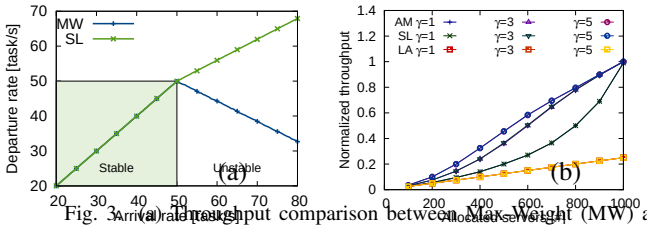

Fig. 3. (a) Throughput comparison between Max-Weight (MW) and Strict-Local (SL) scheduler. (b) Maximum throughput comparison between Analytical Model (AM) Eq. (2), Strict-Local (SL) and Locality-Agnostic (LA) scheduler.

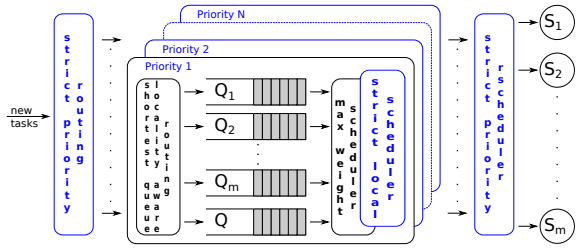

Fig. 4. Scheduler architecture.

batch tasks, higher interactive/batch MapReduce workloads and lower web workloads, the optimal policy is more likely to allocate the whole cluster for MapReduce. However, with less residual batch tasks, lower interactive/batch MapReduce workloads and high web workloads, the optimal policy is more prone to allocate as few servers for MapReduce as possible while ensuring 1) interactive MapReduce tasks are well served and 2) web workload can make use of the residual servers. It is worth noting that in most of the numerical experiments the optimal policy makes extreme decisions similar to "All in or nothing", and rarely falls in between.

IV. Data-locality-aware scheduler

A critical assumption used in our control algorithm described in the previous sections is that the scheduler is able to achieve the optimal throughput function $f(m)$ based on the upper bound probability $p_l(m)$ of finding at least one server having a local copy of the data. To achieve this optimal throughput we consider the scheduler presented in [14], with several critical enhancements. First, we extend the original scheduler with priority classes to handle multiple task classes at the same time. In particular we consider two classes one for batch (low priority) and one for interactive (high priority) MapReduce. Second, we address a stability issue of the original scheduler when it is forced to work in the overload region. This is critical requirement since our system design relies on temporary overloaded regions to delay the low priority batch MapReduce tasks. Third, we optimize the scheduler to deal with time-varying number of servers m_t.

The base version of this scheduler from [14] handles one queue per server S for local tasks[5], denominated $Q_1 \dots Q_m$, plus one for remote tasks, denominated Q, shared across all servers. These queues are handled through two policies: one to route a new task when it arrives to the system and one to schedule the next task to be processed when a server becomes available. Figure 3 shows the original scheduler in black and our enhancements in blue.

The original scheduler uses a join-the-shortest-queue policy to route new tasks to either the shared remote queue or the appropriate available local queue(s), i.e., the queues corresponding to servers having a local copy of the data. When a server i becomes available, the original scheduler uses a max-weight policy which picks the task from either the associeted local queue Q_i or the shared remote queue Q based on the queue lengths weighted by the inverse of the associated slowdown factor, i.e., 1 for the local queue and α for the remote queue. To be able to handle multiple task classes, we extended the original scheduler with a set of such queues $\mathbf{Q}$ for each priority class p in the system. When a new task of priority p arrives, it is first routed to the correponding set of queues $\mathbf{Q}^p$ based on its priority class and then joins the shortest queue as in the base version. When a server i becomes available, the scheduler will first try to dispatch a task from the highest priority queue set $\mathbf{Q}^p$ where $p = P$. If both the local queue Q_i^p and remote queue Q^p are empty the scheduler will consider the next set of queues in order of decreasing priority.

The original data-locality-aware scheduler from [14] performs very poorly in the overload region, i.e., arrival rate greater than departure rate. This can be clearly seen in Figure 4 (a). Our system design however relies on overloaded control periods to delay the batch MapReduce workload. Indeed, during the periods in which the number of available MapReduce servers m is set low, the arrival rate exceeds the maximum available capacity. If m is low, the probability of finding a local copy $P_l(m)$ is also low and most tasks go to the remote queue. The problem lies in the max-weight policy which will mainly dispatch tasks from the remote queue being the longest causing a system throughput lower than the optimal one. To overcome the problem we replace the max-weight policy with a strict-local policy which always serves first the local queue. Without any control this policy bears the danger of starving the remote tasks. In our system the controller has the goal and responsibility to always allocate enough servers to handle the workload over the time horizon. Temporarily "starving", i.e., delaying, the lower priority batch MapReduce tasks, is part of our system design.

For the complete system scenario, we include two more scheduler optimizations to better deal with the transition between different values of m_t. If m_t decreases, servers must be freed up. The scheduler will do so in reverse order of the length of the local queues, i.e., trying to preserve the servers with most local tasks. If m_t increases, new servers are available. The scheduler rebalances the remote queue trying to move tasks from the remote queue to the newly available local queue(s).

To empirically prove that it is possible to achieve the optimal maximum throughput $f(m)$ assumed in Section II, we

[5]For the sake of briefness we refer to tasks accessing local/remote data as local/remote tasks

repeat the simulation shown in Figure 4 (a) while varying the number of available servers. We then evaluate the maximum achievable throughput by the position of the knee. As a baseline we also consider a simple data-locality-agnostic priority scheduler. The results are shown in Figure 4 (b). One can easily see that the maximum throughput of our enhanced strict-local scheduler basically overlaps with the optimal achievable throughput, both largely outperforming the locality-agnostic scheduler. Hereon, our strict-local scheduler will be the default scheduler.

V. Evaluation

In this section, we use simulation to compare our optimal perfect-knowledge (PK) algorithm with our MDP-based algorithm and 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.

A. 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, depending on the data availability. 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 [21].

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 datacenters. To make the two patterns comparable, both use the same mean arrival rate across the 24 hours: $\mathbb{E}(\lambda_{b,t}) = 20$ and $\mathbb{E}(\lambda_{c,t}) = 5$. All inter-arrival rates and task execution times are exponentially distributed with mean $\frac{1}{\lambda}$ and $\mu = 10$s, respectively, Finally, each task is randomly assigned to a data chunk which is uniformly distributed across the MapReduce servers with replication factor $\gamma = [1,3,5]$.

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.

B. 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 different algorithms on the energy consumption in Figure 5 (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 schedulers outperform the baseline by up to 59.3% and the dummy controllers by up to 30.7%. Surprisingly, the MDP controller is not worse than the PK controller in most cases. The PK controller gains from its knowledge of the future, but it suffers more from any randomness-induced deviation due to its open-loop mode of operation. On the contrary MDP controller might suffer from the complexity constrained discretization levels, but can react to system deviations thanks to its closed operation. This is a welcome result since MDP better adapts to practical use cases. Finally, one can observe higher energy savings when increasing the replication factor. A higher replication factor allows to raise the probability of finding a server with a local data copy and the achievable throughput. Hence it allows to diminish the number of servers required to satisfy the same demand.

C. (Un)Finished Tasks and Its Locality

While energy minimization is the optimization objective, we still want to satisfy all the requests coming to the system. Considering again all four controllers and the same scenarios as in our previous section, we depict in Figure 6 (a) and (b) the amount of finished tasks over the whole time horizon split per data locality and type and compare it to the total input load. One can observe that both the MDP and D1 controllers are able to almost always complete all the tasks, while PK and, definitely, D2 lag behind. Here the PK controller really 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 both D1 and MDP controllers succeed, D1 lacks the energy saving possibilities given by the MDP controller. Hence, MDP consistently achieves the best 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.

Looking more in detail and comparing the number of interactive and batch MapReduce tasks, we see that it mostly reflects the 1 to 4 ratio between the two classes input lambdas. Only when there are unfinished task this ratio tilts in favor

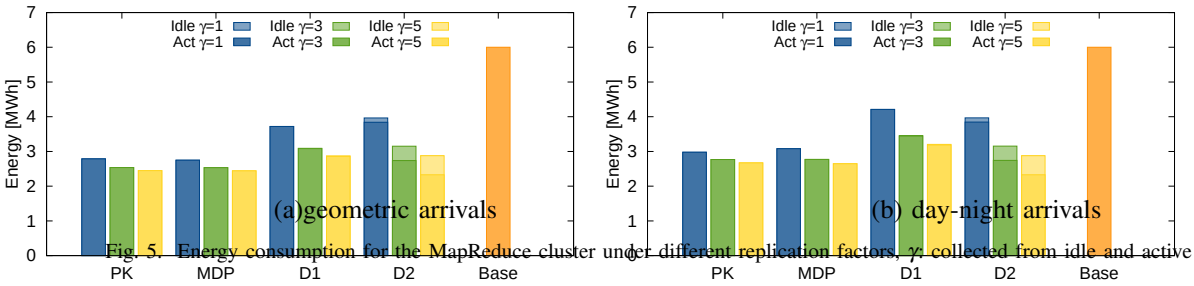

Fig. 5. Energy consumption for the MapReduce cluster under different replication factors, γ, collected from idle and active servers

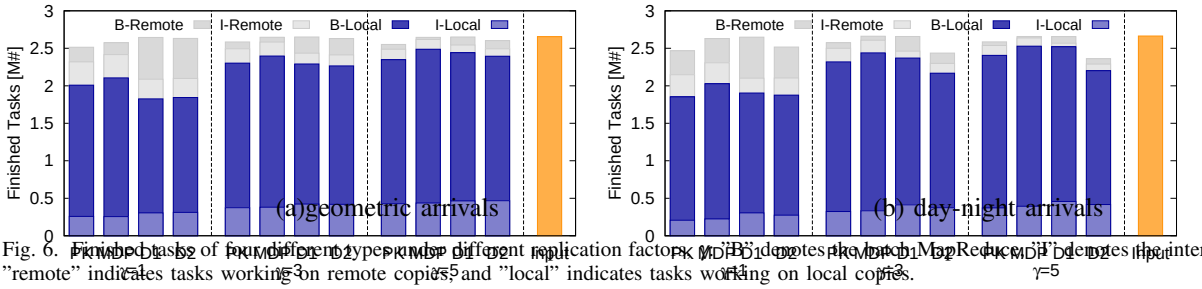

Fig. 6. Finished tasks of four different types under different replication factors, γ. "B" denotes the batch MapReduce, "I" denotes the interaction MapReduce, "remote" indicates tasks working on remote copies, and "local" indicates tasks working on local copies.

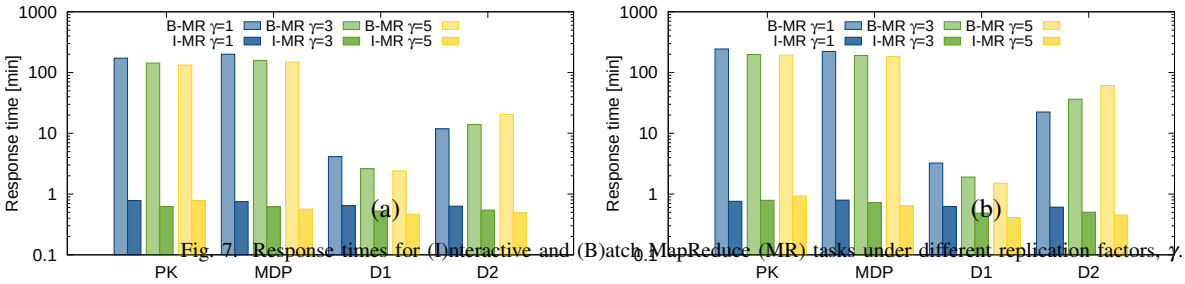

Fig. 7. Response times for (I)nteractive and (B)atch MapReduce (MR) tasks under different replication factors, γ.

of the interactive MapReduce due to their higher priority. Similarly comparing the amount of local and remote tasks we observe again the influence of the replication factor. With a $\gamma = 1$ the average local task are 75% which increases to 89% with $\gamma = 3$ and 94% with $\gamma = 5$.

D. Response Times

We conclude our evaluation presenting the effect of each control allocation on the mean response time. These are shown in Figure 7 (a) and (b) split by interactive and batch MapReduce. Comparing across the two MapReduce classes, we can easily confim that the priority scheduler is able to allocate more resources to interactive MapReduce. This results in lower response times fro interactive MapReduce across all scenarios even if both classes have the same service demand. Comparing across the four controllers, the best results are obtained by D1, followed by D2. Overall the energy minimizing controllers aim at delaying part of the workload to be able to group it and achieve higher energy efficiency. Hence they loose against D1, which tries to always finish all the workload in each control window, and D2 which delays only the peak demands. Even so, PK and MDP succeed in delaying mostly the batch MapReduce tasks while keeping the interactive MapReduce delay almost in par with D1 and D2.

VI. Related Work

There are plethora of studies aiming to improve energy efficiency for different types of systems, ranging from conventional web service systems [**?**], [**?**], [7], [22] to modern big data systems [3], [11], [13], [23], 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 and scheduling policies, with special focus on MapReduce systems.

A. Dynamic Sizing

Dynamic sizing is proven effective for workloads that show strong time varying patterns, e.g., switching on/off servers for web services [7], [22]. Due to the issues of data accessibility of underlying file systems [12] and performance dependency on data localitydynamic 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 [13]. On the one hand, to harvest the maximum data locality, data intensive workloads are batched and executed together on the MapReduce clusters [3], [11] for certain duration, often at midnight. On the other hand, another set of studies [3], [13] leverage the data replication factor (e.g., 3 replicas per data [10]) 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 [3] 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 [16] try to schedule MapReduce workloads according to the dynamics of web workloads, considering only the batch MapReduce and (often) overlooking the locality issues. In our prior work [20], we optimized the dynamic allocation of servers to web and MapReduce workloads without considering workload uncertainty and locality aware scheduling. All in all, the related work partially address how to dynamically size the server resources for executing web, interactive and batch MapReduce in an energy optimal fashion.

B. Scheduler

The scheduler design is one of central themes for MapReduce. The majority of related work centers on the performance of particular types, either through the system development or theoretical framework. A few studies [25] explore the opportunities of energy optimization for MapReduce, only using the scheduler without controlling server allocation. On the one hand, several practical schedulers, in particular priority ones, are developed for various types of big data applications, e.g., Sparks [4] on stream processing, Natjam [5] on batch, and BlinkDB [6] on interactive queries. Wood [16] developed hybrid scheduler for executing MapReduce and service workloads. On the other hand, Tan et. al. [14] develop scheduling algorithms, which are shown optimal in achieving perfect data locality and thus throughput. However, they consider single type of MapReduce in a stationary environment with constant number of servers and where the system loads never saturate. 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.

VII. 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 $O(T^2)$ and relaxed our assumption via MDP to model a certain length of optimization horizon, e.g., one day. We contrived a throughput optimal scheduler which improves upon the state of the art via throughput stability in the overload region and task prioritization support. 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.

Appendix

First, we need following definitions and lemmas.

Definition 3 (Submodularity): Let X and Y be partially ordered sets and $g(x,y)$ a real-valued function on $X \times Y$. We say that g is submodular if for $x^+ \geq x^-$ in X and $y^+ \geq y^-$ in Y, it holds that

$$g(x^+,y^+) + g(x^-,y^-) \leq g(x^+,y^-) + g(x^-,y^+) \,.$$

Moreover, if $X = Y = \mathbb{R}$ and $g(x,y)$ is twice differentiable, then submodularity is equivalent to

$$\frac{\partial^2 g(x,y)}{\partial x \, \partial y} \geq 0 \,.$$

Lemma 4: Suppose g is a submodular function on $X \times Y$ and for each $x \in X$, $\min_{y \in Y} g(x,y)$ exists. Then

$$f(x) = \min \left\{ y' \in \operatorname*{argmin}_{y \in Y} g(x,y) \right\} \,.$$

is monotone non-decreasing in x.

Lemma 5: Let $\{x_j\}$, $\{x'_j\}$ be real-valued non-negative sequences satisfying

$$\sum_{j=k}^{\infty} x_j \geq \sum_{j=k}^{\infty} x'_j \,,$$

for all k, with equality holding for $k = 0$. Suppose $v_{j+1} \geq v_j$ for $j = 0, 1, \ldots$, then

$$\sum_{j=0}^{\infty} v_j x_j \geq \sum_{j=0}^{\infty} v_j x'_j \,.$$

Lemma 6: Assume random variable X is sampled from some geometric distribution $P_X(x) = (1-p)^x p$, it holds that the function

$$f(k) = \sum_{x=k}^{\infty} P_X(x)$$

is a decreasing discrete convex sequence in k. Namely, for any $k_1 \leq k_2 \leq k_3 \leq k_4$ satisfying $k_1 + k_4 \leq k_2 + k_3$, it holds that

$$f(k_1) + f(k_4) \geq f(k_2) + f(k_3) \,.$$

We have $f(k)=\sum_{x=k}^{\infty}(1-p)^x p=(1-p)^k$, which can be easily verified to be a decreasing discrete convex sequence.

A. Proof for Theorem 8

We first prove that if $\lambda_{b,t}$ are sampled independently from some geometric distribution (or exponential distribution), there exists an optimal policy increasing in $r_t+\tau\cdot\lambda_{b,t}$.[6] The state transition probability is

$$\begin{aligned}&Pr(s_{t+1}|s_t,m_t)=\\&Pr(r_{t+1}+\tau\lambda_{b,t+1},\lambda_{c,t+1},l_{t+1}|r_t+\tau\lambda_{b,t},\lambda_{c,t},l_t,m_t)=\\&P_b((r_{t+1}+\tau\lambda_{b,t+1}-(r_t+\tau\lambda_{b,t}+\tau\lambda_{c,t}-f(m_t))^+)/\tau)\cdot\\&P_c(\lambda_{c,t+1})\cdot P_l(l_{t+1})\,,\end{aligned}\tag{8}$$

where $P_b(\cdot)$, $P_c(\cdot)$ and $P_l(\cdot)$ are the probabilities for random variables $\lambda_{b,t}$, $\lambda_{c,t}$ and l_t (derived from $\lambda_{w,t}$), respectively.

We use s_t^+ and s_t^- to denote $((r_t+\tau\lambda_{b,t})^+,\lambda_{c,t},l_t)$ and $((r_t+\tau\lambda_{b,t})^-,\lambda_{c,t},l_t)$, where $(r_t+\tau\lambda_{b,t})^+\geq(r_t+\tau\lambda_{b,t})^-$. We use $f(m_t^+)$ and $f(m_t^-)$ to denote arbitrary two feasible actions, where $m_t^+\geq m_t^-$. Define that

$$\begin{aligned}x^{++}&=\left(r_{t+1}+\tau\lambda_{b+1,t}-((r_t+\tau\lambda_{b,t})^++\tau\lambda_{c,t}-f(m_t^+))^+\right)/\tau\\x^{+-}&=\left(r_{t+1}+\tau\lambda_{b+1,t}-((r_t+\tau\lambda_{b,t})^++\tau\lambda_{c,t}-f(m_t^-))^+\right)/\tau\\x^{-+}&=\left(r_{t+1}+\tau\lambda_{b+1,t}-((r_t+\tau\lambda_{b,t})^-+\tau\lambda_{c,t}-f(m_t^+))^+\right)/\tau\\x^{--}&=\left(r_{t+1}+\tau\lambda_{b+1,t}-((r_t+\tau\lambda_{b,t})^-+\tau\lambda_{c,t}-f(m_t^-))^+\right)/\tau\,.\end{aligned}$$

It holds that $x^{+-}\leq x^{--}\leq x^{-+}$, $x^{+-}\leq x^{++}\leq x^{-+}$ and $x^{+-}+x^{-+}\leq x^{--}+x^{++}$. By Lemma 13, we have

$$\sum_{x=x^{++}}^{\infty}P_b(x)+\sum_{x=x^{--}}^{\infty}P_b(x)\leq\sum_{x=x^{+-}}^{\infty}P_b(x)+\sum_{x=x^{-+}}^{\infty}P_b(x)\,.\tag{9}$$

Putting (10) and (11) together, it follows that the function

$$\begin{aligned}&\sum_{r_{t+1}+\tau\lambda_{b,t+1}=k}^{\infty}Pr(s_{t+1}|s_t,m_t)\;:=\\&\sum_{r_{t+1}+\tau\lambda_{b,t+1}=k}^{\infty}Pr(r_{t+1}+\tau\lambda_{b,t+1},\lambda_{c,t+1},l_{t+1}|r_t+\tau\lambda_{b,t},\lambda_{c,t},l_t,m_t)\end{aligned}$$

is submodular in $r_t+\tau\lambda_{b,t}$ and m_t.

It can be proved by induction that $u_{t+1}^*(s_{t+1})$ is non-decreasing in $r_{t+1}+\tau\lambda_{b,t+1}$. Applying Lemma 12 yields

$$\begin{aligned}&\sum_{r_{t+1}+\tau\lambda_{b,t+1}=0}^{\infty}[Pr(s_{t+1}|s_t^+,m_t^+)+Pr(s_{t+1}|s_t^-,m_t^-)]u_{t+1}^*(s_{t+1})\\\leq&\sum_{r_{t+1}+\tau\lambda_{b,t+1}=0}^{\infty}[Pr(s_{t+1}|s_t^-,m_t^+)+Pr(s_{t+1}|s_t^+,m_t^-)]u_{t+1}^*(s_{t+1})\end{aligned}$$

Summing over $\lambda_{c,t+1}$ and l_{t+1}, we get

$$\begin{aligned}&\mathbb{E}(u_{t+1}^*(s_{t+1})|s_t^+,m_t^+)+\mathbb{E}(u_{t+1}^*(s_{t+1})|s_t^-,m_t^-)\\\leq&\,\mathbb{E}(u_{t+1}^*(s_{t+1})|s_t^+,m_t^-)+\mathbb{E}(u_{t+1}^*(s_{t+1})|s_t^-,m_t^+)\,.\end{aligned}$$

Thus, $\mathbb{E}(u_{t+1}^*(s_{t+1})|s_t,m_t)$ is submodular in b_t and a_t. Since $K\cdot m_t$ is a single variable function in m_t, it follows that

$$O(s_t,m_t)=K\cdot m_t+\mathop{\mathbb{E}}_{s_{t+1}}(u_{t+1}^*(s_{t+1})|s_t,m_t)$$

is a submodular function in $r_t+\tau\lambda_{b,t}$ and m_t. The result follows from Lemma 11.

[6]This implies the optimal policy is monotone in both r_t and $\lambda_{b,t}$.

B. Proof for Theorem 9

Since l_t is derived from $l_t=f(M-m_t')$, while m_t' is strictly increasing in $\lambda_{c,t}$, it suffices to prove that there exists an optimal solution decreasing in l_t. Moreover, l_t has no effects on the value of the function

$$O(s_t,m_t)=K\cdot m_t+\mathop{\mathbb{E}}_{s_{t+1}}(u_{t+1}^*(s_{t+1})|s_t,m_t)\,.$$

It only affects the feasible region of m_t. More precisely, the feasible set is $\mathbb{M}_t=\{m_t|\max\{\tau\lambda_{c,t},l_t\}\leq f(m_t)\leq A\}$. The upper bound is fixed and the lower bound is non-decreasing in l_t, which implies

$$m_t^*(s_t)=\mathop{\operatorname{argmin}}_{m_t\in\mathbb{M}_t}(O(s_t,m_t))$$

is non-decreasing in l_t.

Then the existence of optimal policy increasing in $\lambda_{c,t}$ follows, since mathematically, increasing $\lambda_{c,t}$ is equivalent to increasing both $\lambda_{b,t}$ and l_t.

REFERENCES

[1] "Hadoop," http://hadoop.apache.org.

[2] T. Condie, N. Conway, P. Alvaro, J. M. Hellerstein, K. Elmeleegy, and R. Sears, "MapReduce Online," in *NSDI*, 2010, pp. 313–328.

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

[4] 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*, 2012, pp. 2–2.

[5] 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*, 2013, pp. 6:1–6:17.

[6] 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*, 2013, pp. 29–42.

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

[8] 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*, 2012, pp. 175–186.

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

[10] "How Google Works," http://www.baselinemag.com/c/a/Infrastructure/How-Google-Works-1/.

[11] W. Lang and J. M. Patel, "Energy Management for MapReduce Clusters," *PVLDB*, vol. 3, no. 1, pp. 129–139, 2010.

[12] "Hadoop HDFS," http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html.

[13] J. Leverich and C. Kozyrakis, "On the energy (in)efficiency of Hadoop clusters," *Operating Systems Review*, vol. 44, no. 1, pp. 61–65, 2010.

[14] W. Wang, K. Zhu, L. Ying, J. Tan, and L. Zhang, "Map task scheduling in mapreduce with data locality: Throughput and heavy-traffic optimality," in *INFOCOM*, 2013, pp. 1609–1617.

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

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

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

[18] R. Horst, "On the global minimization of concave functions," *Operations Research Spektrum*, vol. 6, no. 4, pp. 195–205, 1984.

[19] P. M. Pardalos and J. B. Rosen, "Methods for global concave minimization: A bibliographic survey," *Siam Review*, vol. 28, no. 3, pp. 367–379, 1986.

[20] Y. Ying, R. Birke, C. Wang, L. Y. Chen, and G. Natarajan, "On energy-aware allocation and execution for batch and interactive mapreduce," *ACM Performance Evaluation Review*, to appear.

[21] 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*. ACM, 2014, pp. 177–192.

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

[23] T. Wirtz and R. Ge, "Improving mapreduce energy efficiency for computation intensive workloads," in *IGCC*, 2011, pp. 1–8.

[24] N. Maheshwari, R. Nanduri, and V. Varma, "Dynamic energy efficient data placement and cluster reconfiguration algorithm for mapreduce framework," *Future Generation Comp. Syst.*, vol. 28, no. 1, pp. 119–127, 2012.

[25] F. Yan, L. Cherkasova, Z. Zhang, and E. Smirni, "Optimizing power and performance trade-offs of mapreduce job processing with heterogeneous multi-core processors," in *Cloud*, 2014.

[26] J. Tan, X. Meng, and L. Zhang, "Coupling task progress for mapreduce resource-aware scheduling," in *INFOCOM*, 2013, pp. 1618–1626.