Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis

Yanpei Chen, Sara Alspaugh, Dhruba Borthakur*, Randy Katz

University of California, Berkeley, *Facebook

(ychen2, alspaugh, randy)@eecs.berkeley.edu, dhruba@fb.com

Abstract

MapReduce workloads have evolved to include increasing amounts of time-sensitive, interactive data analysis; we refer to such workloads as *MapReduce with Interactive Analysis* (MIA). Such workloads run on large clusters, whose size and cost make energy efficiency a critical concern. Prior works on MapReduce energy efficiency have not yet considered this workload class. Increasing hardware utilization helps improve efficiency, but is challenging to achieve for MIA workloads. These concerns lead us to develop BEEMR (Berkeley Energy Efficient MapReduce), an energy efficient MapReduce workload manager motivated by empirical analysis of real-life MIA traces at Facebook. The key insight is that although MIA clusters host huge data volumes, the interactive jobs operate on a small fraction of the data, and thus can be served by a small pool of dedicated machines; the less time-sensitive jobs can run on the rest of the cluster in a batch fashion. BEEMR achieves 40-50% energy savings under tight design constraints, and represents a first step towards improving energy efficiency for an increasingly important class of datacenter workloads.

Categories and Subject Descriptors D.4.7 [*Organization and Design*]: Distributed systems; D.4.8 [*Performance*]: Operational analysis

Keywords MapReduce, energy efficiency.

1. Introduction

Massive computing clusters are increasingly being used for data analysis. The sheer scale and cost of these clusters make it critical to improve their operating efficiency, including energy. Energy costs are a large fraction of the total cost of ownership of datacenters [6, 24]. Consequently, there is a concerted effort to improve energy efficiency for Internet datacenters, encompassing government reports [52], standardization efforts [50], and research projects in both industry and academia [7, 16, 19, 27–29, 32, 33, 43, 48].

Approaches to increasing datacenter energy efficiency depend on the workload in question. One option is to increase machine utilization, i.e., increase the amount of work done per unit energy. This approach is favored by large web search companies such as Google, whose machines have persistently low utilization and waste considerable energy [5]. Clusters implementing this approach would service a mix of interactive and batch workloads [14, 35, 40], with the interactive services handling the external customer queries [32], and batch processing building the data structures that support the interactive services [15]. This strategy relies on predictable diurnal patterns in web query workloads, using latency-insensitive batch processing drawn from an "infinite queue of low-priority work" to smooth out diurnal variations, to keep machines at high utilization [5, 14, 40].

This paper focuses on an alternate use case—what we call *MapReduce with Interactive Analysis (MIA)* workloads. MIA workloads contain interactive services, traditional batch processing, and large-scale, latency-sensitive processing. The last component arises from human data analysts interactively exploring large data sets via ad-hoc queries, and subsequently issuing large-scale processing requests once they find a good way to extract value from the data [9, 26, 34, 54]. Such human-initiated requests have flexible but not indefinite execution deadlines.

MIA workloads require a very different approach to energy-efficiency, one that focuses on decreasing the amount of energy used to service the workload. As we will show by analyzing traces of a front-line MIA cluster at Facebook, such workloads have arrival patterns beyond the system's control. This makes MIA workloads unpredictable: new data sets, new types of processing, and new hardware are added rapidly over time, as analysts collect new data and discover new ways to analyze existing data [9, 26, 34, 54]. Thus, increasing utilization is insufficient: First, the workload is dominated by human-initiated jobs. Hence, the cluster must be provisioned for peak load to maintain good SLOs, and low-priority batch jobs only partially smooth out the workload variation. Second, the workload has unpredictable high spikes compared with regular diurnal patterns for web queries, resulting in wasted work from batch jobs being preempted upon sudden spikes in the workload.

EuroSys'12, April 10–13, 2012, Bern, Switzerland.

MIA-style workloads have already appeared in several organizations, including both web search and other businesses [9, 26, 34]. Several technology trends help increase the popularity and generality of MIA workloads:

- Industries ranging from e-commerce, finance, and manufacturing are increasingly adopting MapReduce as a data processing and archival system [23].
- It is increasingly easy to collect and store large amounts of data about both virtual and physical systems [9, 17, 27].
- Data analysts are gaining expertise using MapReduce to process big data sets interactively for real-time analytics, event monitoring, and stream processing [9, 26, 34].

In short, MapReduce has evolved far beyond its original use case of high-throughput batch processing in support of web search-centric services, and it is critical that we develop energy efficiency mechanisms for MIA workloads.

This paper presents BEEMR (Berkeley Energy Efficient MapReduce), an energy efficient MapReduce system motivated by an empirical analysis of a real-life MIA workload at Facebook. This workload requires BEEMR to meet stringent design requirements, including minimal impact on interactive job latency, write bandwidth, write capacity, memory set size, and data locality, as well as compatibility with distributed file system fault tolerance using error correction codes rather than replication. BEEMR represents a new design point that combines batching [28], zoning [29], and data placement [27] with new analysis-driven insights to create an efficient MapReduce system that saves energy while meeting these design requirements. The key insight is that although MIA clusters host huge volumes of data, the interactive jobs operate on just a small fraction of the data, and thus can be served by a small pool of dedicated machines; whereas the less time-sensitive jobs can run in a batch fashion on the rest of the cluster. These defining characteristics of MIA workloads both motivate and enable the BEEMR design. BEEMR increases cluster utilization while batches are actively run, and decreases energy waste between batches because only the dedicated interactive machines need to be kept at full power. The contributions of this paper are:

- An analysis of a Facebook cluster trace to quantify the empirical behavior of a MIA workload.
- The BEEMR framework which combines novel ideas with existing MapReduce energy efficiency mechanisms.
- An improved evaluation methodology to quantify energy savings and account the complexity of MIA workloads.
- An identification of a set of general MapReduce design issues that warrant more study.

We show energy savings of 40-50%. BEEMR highlights the need to design for an important class of data center workloads, and represents an advance over existing MapReduce energy efficiency proposals [27–29]. Systems like BEEMR become more important as the need for energy efficiency continues to increase, and more use cases approach the scale and complexity of the Facebook MIA workload.

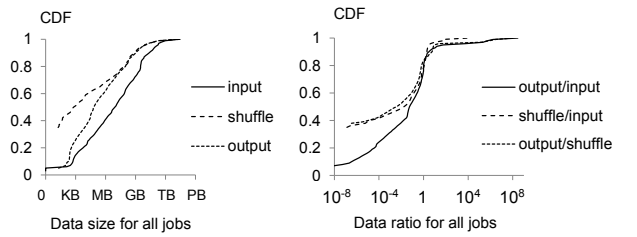


Figure 1. CDFs of input/shuffle/output sizes and ratios for the entire 45-day Facebook trace. Both span several orders of magnitudes. Energy efficiency mechanisms must accommodate this range.

2. Motivation

Facebook is a social network company that allows users to create profiles and connect with each other. The Facebook workload provides a detailed case study of the growing class of MIA workloads. This analysis motivates the BEEMR design and highlights where previous solutions fall short.

2.1 The Facebook Workload

We analyze traces from the primary Facebook production Hadoop cluster. The cluster has 3000 machines. Each machine has 12+ TB, 8-16 cores, 32 GB of RAM, and roughly 15 concurrent map/reduce tasks [8]. The traces cover 45 days from Oct. 1 to Nov. 15, 2010, and contain over 1 million jobs touching tens of PB of data. The traces record each job's job ID, input/shuffle/output sizes, arrival time, duration, map/reduce task durations (in task-seconds), number of map/reduce tasks, and input file path.

Figure 1 shows the distribution of per-job data sizes and data ratios for the entire workload. The data sizes span several orders of magnitude, and most jobs have data sizes in the KB to GB range. The data ratios also span several orders of magnitude. 30% of the jobs are map-only, and thus have 0 shuffle data. Any effort to improve energy efficiency must account for this range of data sizes and data ratios.

Figure 2 shows the workload variation over two weeks. The number of jobs is diurnal, with peaks around midday and troughs around midnight. All three time series have a high peak-to-average ratio, especially map and reduce task times. Since most hardware is not power proportional [5], a cluster provisioned for peak load would see many periods of below peak activity running at near-peak power.

To distinguish among different types of jobs in the workload, we can perform statistical data clustering analysis. This analysis treats each job as a multi-dimensional vector, and finds clusters of similar numerical vectors, i.e., similar jobs. Our traces give us six numerical dimensions per job — input size, shuffle size, output size, job duration, map time, and reduce time. Table 1 shows the results using the k-means algorithm, in which we labeled each cluster based on the numerical value of the cluster center.

Most of the jobs are small and interactive. These jobs arise out of ad-hoc queries initiated by internal human analysts at Facebook [9, 51]. There are also jobs with long du-

# Jobs	Input	Shuffle	Output	Duration	Map time	Reduce time	Label
1145663	6.9 MB	600 B	60 KB	1 min	48	34	Small jobs
7911	50 GB	0	61 GB	8 hrs	60,664	0	Map only transform, 8 hrs
779	3.6 TB	0	4.4 TB	45 min	3,081,710	0	Map only transform, 45 min
670	2.1 TB	0	2.7 GB	1 hr 20 min	9,457,592	0	Map only aggregate
104	35 GB	0	3.5 GB	3 days	198,436	0	Map only transform, 3 days
11491	1.5 TB	30 GB	2.2 GB	30 min	1,112,765	387,191	Aggregate
1876	711 GB	2.6 TB	860 GB	2 hrs	1,618,792	2,056,439	Transform, 2 hrs
454	9.0 TB	1.5 TB	1.2 TB	1 hr	1,795,682	818,344	Aggregate and transform
169	2.7 TB	12 TB	260 GB	2 hrs 7 min	2,862,726	3,091,678	Expand and aggregate
67	630 GB	1.2 TB	140 GB	18 hrs	1,545,220	18,144,174	Transform, 18 hrs

Table 1. Job types in the workload as identified by k-means clustering, with cluster sizes, medians, and labels. Map and reduce time are in task-seconds, i.e., a job with 2 map tasks of 10 seconds each has map time of 20 task-seconds. Notable job types include small, interactive jobs (top row) and jobs with inherently low levels of parallelism that take a long time to complete (fifth row). We ran k-means with 100 random instantiations of cluster centers, which averages to over 1 bit of randomness in each of the 6 data dimensions. We determine k, the number of clusters by incrementing k from 1 and stopping upon diminishing decreases in the intra-cluster "residual" variance.

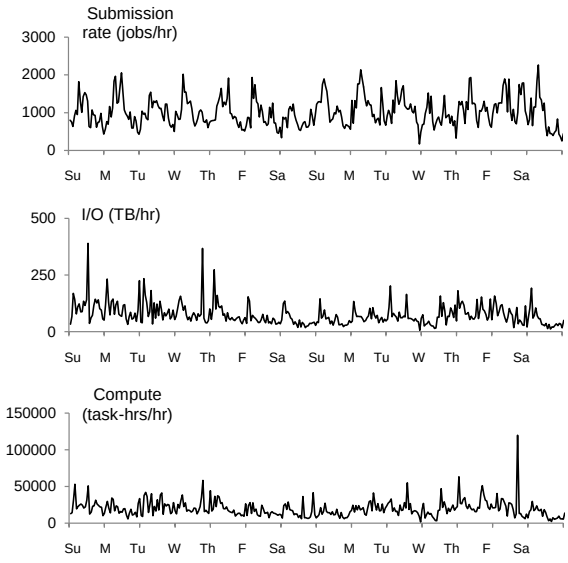


Figure 2. Hourly workload variation over two weeks. The workload has high peak-to-average ratios. A cluster provisioned for the peak would be often underutilized and waste a great deal of energy.

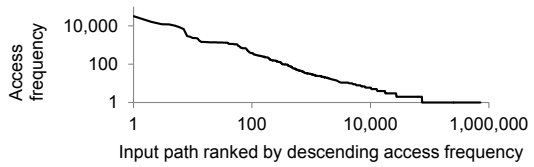


Figure 3. Log-log plot of workload input file path access frequency. This displays a Zipf distribution, meaning that a few input paths account for a large fraction of all job inputs.

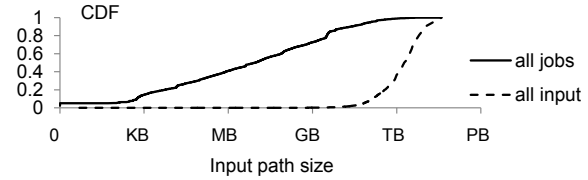


Figure 4. CDF of both (1) the input size per job and (2) the size per input path. This graph indicates that small input paths are accessed frequently, i.e., data sets of less than 10s of GBs account for over 80% of jobs, and such data sets are a tiny fraction of the total data stored on the cluster.

rations but small task times (map only, GB-scale, many-day jobs). These jobs have inherently low levels of parallelism, and take a long time to complete, even if they have the entire cluster at their disposal. Any energy efficient MapReduce system must accommodate many job types, each with their own unique characteristics.

Figures 3 and 4 show the data access patterns as indicated by the per-job input paths. Unfortunately our traces do not contain comparable information for output paths. Figure 3 shows that the input path accesses follow a Zipf distribution, i.e., a few input paths account for a large fraction of all accesses. Figure 4 shows that small data sets are accessed frequently; input paths of less than 10s of GBs account for over 80% of jobs, but only a tiny fraction of the total size of all input paths. Prior work has also observed this behavior in other contexts, such as web caches [10] and databases [21]. The implication is that *a small fraction of the cluster is sufficient to store the input data sets of most jobs.*

Other relevant design considerations are not evident from the traces. First, some applications require high write throughput and considerable application-level cache, such as Memcached. This fact was reported by Facebook in [9] and [51]. Second, the cluster is storage capacity constrained, so Facebook's HDFS achieves fault tolerance through error correcting codes instead of replication, which brings the physical replication factor down from three to less than two [45]. Further, any data hot spots or decreased data locality would increase MapReduce job completion times [2].

Table 2 summarizes the design constraints. They represent a superset of the requirements considered by existing energy efficient MapReduce proposals.

2.2 Prior Work

Prior work includes both energy-efficient MapReduce schemes as well as strategies that apply to other workloads.

2.2.1 Energy Efficient MapReduce

Existing energy efficient MapReduce systems fail to meet all the requirements in Table 2. We review them here.

The covering subset scheme [29] keeps one replica of every block within a small subset of machines called the covering subset. This subset remains fully powered to preserve data availability while the rest is powered down. Operating only a fraction of the cluster decreases write bandwidth, write capacity, and the size of available memory. More critically, this scheme becomes unusable when error correction codes are used instead of replication, since the covering subset becomes the whole cluster.

The all-in strategy [28] powers down the entire cluster during periods of inactivity, and runs at full capacity otherwise. Figure 2 shows that the cluster is never completely inactive. Thus, to power down at any point, the all-in strategy must run incoming jobs in regular batches, an approach we investigated in [13]. All jobs would experience some delay, an inappropriate behavior for the small, interactive jobs in the MIA workload (Table 1).

Green HDFS [27] partitions HDFS into disjoint hot and cold zones. The frequently accessed data is placed in the hot zone, which is always powered. To preserve write capacity, Green HDFS fills the cold zone using one powered-on machine at a time. This scheme is problematic because the output of every job would be located on a small number of machines, creating a severe data hotspot for future accesses. Furthermore, running the cluster at partial capacity decreases the available write bandwidth and memory.

The prior studies in Table 2 also suffer from several methodological weaknesses. Some studies quantified energy efficiency improvements by running stand-alone jobs, similar to [43]. This is the correct initial approach, but it is not clear that improvements from stand-alone jobs translate to workloads with complex interference between concurrent jobs. More critically, for workloads with high peak-to-average load (Figure 2), per-job improvements fail to eliminate energy waste during low activity periods.

Other studies quantified energy improvements using trace-driven simulations. Such simulations are essential for evaluating energy efficient MapReduce at large scale. However, the simulators used there were not empirically verified, i.e., there were no experiments comparing simulated versus real behavior, nor simulated versus real energy savings. Section 5.8 demonstrates that an empirical validation reveals many subtle assumptions about simulators, and put into doubt the results derived from unverified simulators.

These shortcomings necessitate a new approach in designing and evaluating energy efficient MapReduce systems.

2.2.2 Energy Efficient Web Search-Centric Workloads

MIA workloads require a different approach to energy efficiency than previously considered workloads.

In web search-centric workloads, the interactive services achieves low latency by using data structures in-memory, requiring the entire memory set to be always available [32]. Given hardware limits in power proportionality, it becomes a priority to increase utilization of machines during diurnal troughs [5]. One way to do this is to admit batch processing to consume any available resource. This policy makes the combined workload *closed-loop*, i.e., the system controls the amount of admitted work. Further, the combined workload becomes more *predictable*, since the interactive services display regular diurnal patterns, and with batch processing smoothing out most diurnal variations [5, 19, 32].

These characteristics enable energy efficiency improvements to focus on *maximizing the amount of work done subject to the given power budget*, i.e., maximizing the amount of batch processing done by the system. Idleness is viewed as waste. Opportunities to save energy occur at short time scales, and requires advances in hardware energy efficiency and power proportionality [5, 7, 16, 19, 32, 33, 48].

These techniques remain helpful for MIA workloads. However, the open-loop and unpredictable nature of MIA workloads necessitates additional approaches. Human initiated jobs have both throughput and latency constraints. Thus, the cluster needs to be provisioned for peak, and idleness is inherent to the workload. Machine-initiated batch jobs can only partially smooth out transient activity peaks. Improving hardware power proportionality helps, but remains a partial solution since state-of-the-art hardware is still far from perfectly power proportional. Thus, absent policies to constrain the human analysts, improving energy efficiency for MIA workloads requires *minimizing the energy needed to service the given amount of work*.

More generally, energy concerns complicate capacity provisioning, a challenging topic with investigations dating back to the time-sharing era [3, 4, 46]. This paper offers a new perspective informed by MIA workloads.

3. BEEMR Architecture

BEEMR is an energy efficient MapReduce workload manager. The key insight is that the interactive jobs can be served by a small pool of dedicated machines with their associated storage, while the less time-sensitive jobs can run in a batch fashion on the rest of the cluster using full computation bandwidth and storage capacity. This setup leads to energy savings and meet all the requirements listed in Table 2.

3.1 Design

The BEEMR cluster architecture is shown in Figure 5. It is similar to a typical Hadoop MapReduce cluster, with important differences in how resources are allocated to jobs.

The cluster is split into disjoint interactive and batch zones. The interactive zone makes up a small, fixed percentage of cluster resources — task slots, memory, disk capacity, network bandwidth, similar to the design in [4]. The interac-

Desirable Property	Covering subset [29]	All-In [28]	Hot & Cold Zones [27]	BEEMR
Does not delay interactive jobs	✔		✔	✔
No impact on write bandwidth		✔		✔
No impact on write capacity		✔	✔	✔
No impact on available memory		✔		✔
Does not introduce data hot spots nor impact data locality	✔	✔		✔
Improvement preserved when using ECC instead of replication		✔	✔	✔
Addresses long running jobs with low parallelism				Partially
Energy savings	9-50%[1]	0-50%[2]	24%[3]	40-50%

Table 2. Required properties for energy-saving techniques for Facebook's MIA workload. Prior proposals are insufficient. Notes: [1] The reported energy savings used an energy model based on linearly extrapolating CPU utilization while running the GridMix throughput benchmark [22] on a 36-node cluster. [2] Reported only relative energy savings compared with the covering subset technique, and for only two artificial jobs (Terasort and Grep) on a 24-node experimental cluster. We recomputed absolute energy savings using the graphs in the paper. [3] Reported simulation based energy *cost* savings, assumed an electricity cost of $0.063/KWh and 80% capacity utilization.

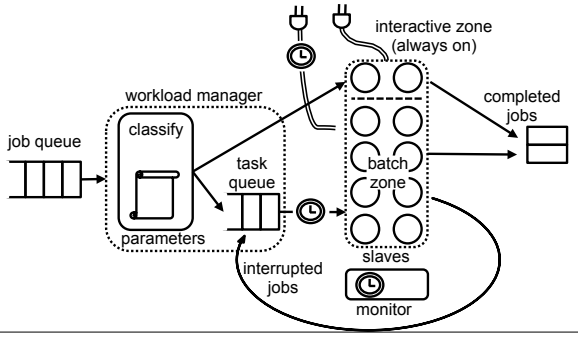


Figure 5. The BEEMR workload manager (i.e., job tracker) classifies each job into one of three classes which determines which cluster zone will service the job. Interactive jobs are serviced in the interactive zone, while batchable and interruptible jobs are serviced in the batch zone. Energy savings come from aggregating jobs in the batch zone to achieve high utilization, executing them in regular batches, and then transitioning machines in the batch zone to a low-power state when the batch completes.

tive zone is always fully powered. The batch zone makes up the rest of the cluster, and is put into a very low power state between batches [25].

As jobs arrive, BEEMR classifies them as one of three job types. Classification is based on empirical parameters derived from the analysis in Section 2. If the job input data size is less than some threshold `interactive`, it is classified as an interactive job. BEEMR seeks to service these jobs with low latency. If a job has tasks with task duration longer than some threshold `interruptible`, it is classified as an interruptible job. Latency is not a concern for these jobs, because their long-running tasks can be check-pointed and resumed over multiple batches. All other jobs are classified as batch jobs. Latency is also not a concern for these jobs, but BEEMR makes best effort to run them by regular deadlines. Such a setup is equivalent to deadline-based policies where the deadlines are the same length as the batch intervals.

The interactive zone is always in a full-power ready state. It runs all of the interactive jobs and holds all of their associated input, shuffle, and output data (both local and HDFS storage). Figures 3 and 4 indicate that choosing an appropriate value for `interactive` can allow most jobs to be classified as interactive and executed without any delay introduced by BEEMR. This `interactive` threshold should be periodically adjusted as workloads evolve.

The interactive zone acts like a data cache. When an interactive job accesses data that is not in the interactive zone (i.e., a cache miss), BEEMR migrates the relevant data from the batch zone to the interactive zone, either immediately or upon the next batch. Since most jobs use small data sets that are reaccessed frequently, cache misses occur infrequently. Also, BEEMR requires storing the ECC parity or replicated blocks within the respective zones, e.g., for data in the interactive zone, their parity or replication blocks would be stored in the interactive zone also.

Upon submission of batched and interruptible jobs, all tasks associated with the job are put in a wait queue. At regular intervals, the workload manager initiates a batch, powers on all machines in the batch zone, and run all tasks on the wait queue using the whole cluster. The machines in the interactive zone are also available for batch and interruptible jobs, but interactive jobs retain priority there. After a batch begins, any batch and interruptible jobs that arrive would wait for the next batch. Once all batch jobs complete, the job tracker assigns no further tasks. Active tasks from interruptible jobs are suspended, and enqueued to be resumed in the next batch. Machines in the batch zone return to a low-power state. If a batch does not complete by start of the next batch interval, the cluster would remain fully powered for consecutive batch periods. The high peak-to-average load in Figure 2 indicates that on average, the batch zone would spend considerable periods in a low-power state.

BEEMR improves over prior batching and zoning schemes by combining both, and uses empirical observations to set the values of policy parameters, which we describe next.

3.1.1 Parameter Space

BEEMR involves several design parameters whose values need to be optimized. These parameters are:

Parameter	Units or Type	Values
`totalsize`	thousand slots	32, 48, 60, 72
`mapreduceratio`	map:reduce slots	$1:1$, $27:14$, (≈ 2.0), $13:5$ (≈ 2.6)
`izonesize`	% total slots	10
`interactive`	GB	10
`interruptible`	hours	6, 12, 24
`batchlen`	hours	1, 2, 6, 12, 24
`taskcalc`	algorithm	default, actual, latency-bound

Table 3. Design space explored. The values for `izonesize` and `interactive` are derived from the analysis in Section 2.1. We scan at least three values for each of the other parameters.

- `totalsize`: the size of the cluster in total (map and reduce) task slots.
- `mapreduceratio`: the ratio of map slots to reduce slots in the cluster.
- `izonesize`: the percentage of the cluster assigned to the interactive zone.
- `interactive`: the input size threshold for classifying jobs as interactive.
- `interruptible`: task duration threshold for classifying jobs as interruptible.
- `batchlen`: the batch interval length.
- `taskcalc`: the algorithm for determining the number of map and reduce tasks to assign to a job.

Table 3 shows the parameter values we will optimize for the Facebook workload. For other workloads, the same tuning process can extract a different set of values. Note that `totalsize` indicates the size of the cluster in units of task slots, which differs from the number machines. One machine can run many task slots, and the appropriate assignment of task slots per machine depends on hardware capabilities.

Another parameter worth further explanation is `taskcalc`, the algorithm for determining the number of map and reduce tasks to assign to a job. An algorithm that provides appropriate task granularity ensures that completion of a given batch is not held up by long-running tasks from some jobs.

BEEMR considers three algorithms: *Default* assigns 1 map per 128 MB of input and 1 reduce per 1 GB of input; this is the default setting in Hadoop. *Actual* assigns the same number of map and reduce tasks as given in the trace and corresponds to settings at Facebook. *Latency-bound* assigns a number of tasks such that no task will run for more than 1 hour. This policy is possible provided that task execution times can be predicted with high accuracy [20, 36].

3.1.2 Requirements Check

We verify that BEEMR meets the requirements in Table 2.

1. Write bandwidth is not diminished because the entire cluster is fully powered when batches execute. Table 1 indicates that only batch and interruptible jobs require large write bandwidth. When these jobs are running, they have access to all of the disks in the cluster.
2. Similarly, write capacity is not diminished because the entire cluster is fully powered on during batches. Between batches, the small output size of interactive jobs (Table 1) means that an appropriate value of `izonesize` allows those job outputs to fit in the interactive zone.
3. The size of available memory also remains intact. The memory of the entire cluster is accessible to batch and interruptible jobs, which potentially have large working sets. For interactive jobs, the default or actual (Facebook) `taskcalc` algorithms will assign few tasks per job, resulting in small in-memory working sets.
4. Interactive jobs are not delayed. The interactive zone is always fully powered, and designated specifically to service interactive jobs without delay.
5. BEEMR spreads data evenly within both zones, and makes no changes that impact data locality. Nonetheless, Figure 3 suggests that there will be some hotspots inherent to the Facebook workload, independent of BEEMR.
6. BEEMR improves energy efficiency via batching. There is no dependence on ECC or replication, thus preserving energy savings regardless of fault tolerance mechanism.
7. Long jobs with low levels of parallelism remain a challenge, even under BEEMR. These jobs are classified as interruptible jobs if their task durations are large, and batch jobs otherwise. If such jobs are classified as batch jobs, they could potentially prevent batches from completing. Their inherent low levels of parallelism cause the batch zone to be poorly utilized when running only these long jobs, resulting in wasted energy. One solution is for experts to label such jobs a priori so that BEEMR can ensure that these jobs are classified as interruptible.

3.2 Implementation

BEEMR involves several extensions to Apache Hadoop.

The job tracker is extended with a wait queue management module. This module holds all incoming batch jobs, moves jobs from the wait queue to the standard scheduler upon each batch start, and places any remaining tasks of interruptible jobs back on the wait queue when batches end. Also, the scheduler's task placement mechanism is modified such that interactive jobs are placed in the interactive zone, and always have first priority to any available slots.

The namenode is modified such that the output of interactive jobs is assigned to the interactive zone, and the output of batch and interruptible jobs is assigned to the batch zone. If either zone approaches storage capacity, it must adjust the fraction of machines in each zone, or expand the cluster.

The Hadoop master is augmented with a mechanism to transfer all slaves in the batch zone in and out of a low-power state, e.g., sending a "hibernate" command via `ssh` and using Wake-on-LAN or related technologies [30]. If batch intervals are on the order of hours, it is acceptable for this transition to complete over seconds or even minutes.

Accommodating interruptible jobs requires a mechanism that can suspend and resume active tasks. The current Hadoop architecture makes it difficult to implement such a mechanism. However, suspend and resume is a key component of fault recovery under Next Generation Hadoop [38]. We can re-purpose for BEEMR those mechanisms.

These extensions will create additional computation and IO at the Hadoop master node. The current Hadoop master has been identified as a scalability bottleneck [47]. Thus, it is important to monitor BEEMR overhead at the Hadoop master to ensure that we do not affect cluster scalability. This overhead would become more acceptable under Next Generation Hadoop, where the Hadoop master functionality would be spread across several machines [38].

4. Evaluation Methodology

The evaluation of our proposed algorithm involves running the Facebook MIA workload both in simulation and on clusters of hundreds of machines on Amazon EC2 [1].

The Facebook workload provides a level of validation not obtainable through stand-alone programs or artificial benchmarks. It is logistically impossible to replay this workload on large clusters at full duration and scale. The high peak to average nature of the workload means that at time scales of less than weeks, there is no way to know whether the results capture transient or average behavior. Enumerating a multi-dimensional design space would also take prohibitively long. Any gradient ascent algorithms are not possible, simply because there is no guarantee that the performance behavior is convex. Combined, these concerns compel us to use experimentally validated simulations.

The simulator is optimized for simulation scale and speed by omitting certain details: job startup and completion overhead, overlapping map and reduce phases, speculative execution and stragglers, data locality, and interference between jobs. This differs from existing MapReduce simulators [37, 53], whose focus on details make it logistically infeasible to simulate large scale, long duration workloads. The simulator assumes a simple, fluid-flow model of job execution, first developed for network simulations as an alternative to packet-level models [31, 42]. There, the motivation was also to gain simulation scale and speed. Section 5.8 demonstrates that the impact on accuracy is acceptable.

Simulated job execution is a function of job submit time (given in the trace), task assignment time (depends on a combination of parameters, including batch length, and number of map and reduce slots), map and reduce execution times (given in the trace), and the number of mappers and reducers chosen by BEEMR (a parameter). Figure 6 shows how the simulator works at a high level.

We empirically validate the simulator by replaying several day-long workloads on a real-life cluster (Section 5.8). This builds confidence that simulation results translate to real clusters. The validation employs previously developed

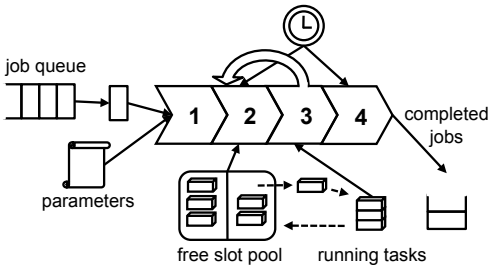


Figure 6. A high-level view of the simulation algorithm. For each simulated second, the following executes: 1. The simulator dequeues newly arrived jobs (arrival pattern given in the trace), classifies the job as interactive, batch, or interruptible, and applies the task granularity policy. 2. The simulator checks for available map or reduce slots, checks the batch policy to see which jobs can be run at the present time, and assigns slots to jobs in round robin, fair scheduler fashion. 3. The simulator removes completed tasks and returns the corresponding slot back to the free slot pool. For each active job, it checks to see if the job has more tasks to run (go back to step 2) or is complete (go to step 4). 4. The job is marked complete and the job duration recorded.

methods to "replay" MapReduce workloads independent of hardware [12]. The techniques there replays the workload using synthetic data sets, and reproduces job submission sequences and intensities, as well as the data ratios between each job's input, shuffle, and output stages.

We model the machines as having "full" power when active, and negligible power when in a low power state. Despite recent advances in power proportionality [5], such models remain valid for Hadoop. In [11], we used wall plug power meters to show that machines with power ranges of 150W-250W draw 205W-225W when running Hadoop. The chattiness of the Hadoop/HDFS stack means that machines are active at the hardware level even when they are idle at the Hadoop workload level. The simple power model allow us to scale the experiments in size and in time.

Several performance metrics are relevant to energy efficient MapReduce: (1) Energy savings: Under our power model, this would be the duration for which the cluster is fully idle; (2) Job latency (analogous to "turn around time" in multiprogramming literature [18]): We measure separately the job latency for each job class, and quantify any trade-off against energy savings; (3) System throughput: Under the MIA open-loop workload model, the historical system throughput would be the smaller of `totalsize` and the historical workload arrival rate. We examine several values of `totalsize` and quantify the interplay between latency, energy savings, and other policy parameters.

Table 3 shows the parameter values used to explore the BEEMR design space.

5. Results

The evaluation spans the multi-dimensional design space in Table 3. Each dimension illustrates subtle interactions between BEEMR and the Facebook workload.

5.1 Cluster Size

Cluster size is controlled by `totalsize`. Underprovisioning a cluster results in long queues and high latency during workload peaks; overprovisioning leads to arbitrarily high baseline energy consumption and waste. Over the 45-days trace, the Facebook workload has an average load of 21029 map tasks and 7745 reduce tasks. Since the workload has a high peak-to-average ratio, we must provision significantly above the average. Figure 7 shows the detailed cluster behavior for several cluster sizes without any of the BEEMR improvements. We pick a one-to-one map-to-reduce-slot ratio because that is the default in Apache Hadoop, and thus forms a good baseline. A cluster with only 32000 total slots cannot service the historical rate, being pegged at maximum slot occupancy; larger sizes still see transient periods of maximum slot occupancy. A cluster with at least 36000 map slots (72000 total slots) is needed to avoid persistent long queues, so we use this as a baseline.

5.2 Batch Interval Length

Energy savings are enabled by batching jobs and transitioning the batch zone to a low-power state between batches. The ability to batch depends on the predominance of interactive analysis in MIA workloads (Section 2.1). We consider here several static batch interval lengths. A natural extension would be to have dynamically adjusted batch intervals to enable various deadline driven policies.

We vary `batchlen`, the batching interval, while holding the other parameters fixed. Figure 8 shows that energy savings, expressed as a fraction of the baseline energy consumption, become non-negligible only for batch lengths of 12 hours or more. Figure 9 shows that map tasks execute in near-ideal batch fashion, with maximum task slot occupancy for a fraction of the batch interval and no further tasks in the remainder of the interval. However, reduce slot occupancy rarely reaches full capacity, while "dangling" reduce tasks often run for a long time at very low cluster utilization. There are more reduce tasks slots available, but the algorithm for choosing the number of task slots limits the amount of parallelism. During the fifth and sixth days, such dangling tasks cause the batch zone to remain at full power for the entire batch interval. Fixing this requires improving both the algorithm for calculating the number of tasks for each job and the ratio of map-to-reduce slots.

5.3 Task Slots Per Job

The evaluation thus far considered only the default algorithm for computing the number of tasks per job, as specified by `taskcalc`. Recall that we consider two other algorithms: *Actual* assigns the same number of map and reduce tasks as given in the trace and corresponds to settings at Facebook. *Latency-bound* assigns a number of tasks such that no task will run for more than 1 hour. Figure 10 compares the default versus actual and latency-bound algorithms. The actual

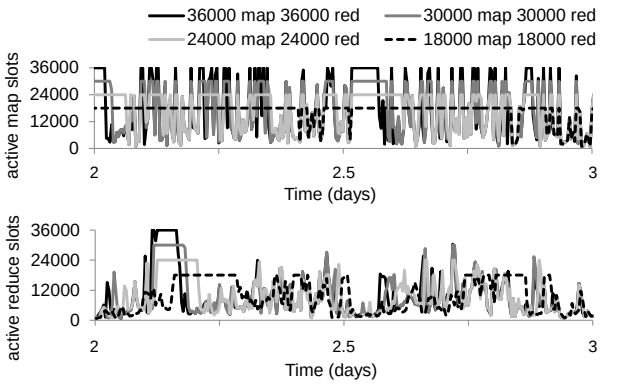


Figure 7. The number of concurrently active tasks for clusters of different sizes (in terms of total task slots, `totalsize`).

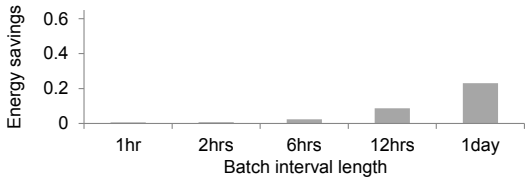


Figure 8. Energy savings for different batch interval lengths as given by `batchlen`. Energy savings are non-negligible for large batch intervals only. Note that `taskcalc` is set to default, `mapreduceratio` is set to 1:1, `totalsize` is set to 72000 slots, and `interruptible` is set to 24 hours.

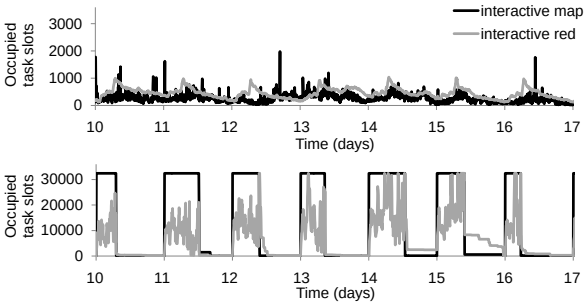


Figure 9. Active slots for a `batchlen` of 24 hours. Showing slot occupancy in the interactive zone (top) and in the batch zone (bottom). Showing one week's behavior. Note that `taskcalc` is set to default, `mapreduceratio` is set to 1:1, `totalsize` is set to 72000 slots, and `interruptible` is set to 24 hours.

policy does the worst, unsurprising because the task assignment algorithm at Facebook is not yet optimized for energy efficiency. The latency-bound policy does the best; this indicates that good task execution time prediction can improve task assignment and achieve greater energy savings.

Observing task slot occupancy over time provides insight into the effects of `taskcalc`. Using the actual algorithm (Figure 11(a)), slots in the interactive zone reach capacity more frequently, suggesting that the Facebook algorithm seeks to increase parallelism to decrease the amount of computation per task and lower the completion latency of interactive jobs. In contrast, tasks in the batch zone behave similarly under the default and Facebook algorithm for

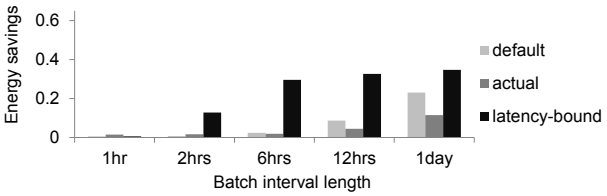


Figure 10. Energy savings for different `taskcalc` algorithms. Note that `mapreduceratio` is set to 1:1, and `interruptible` is set to 24 hours. The *actual* (Facebook) algorithm does worst and the *latency-bound* algorithm does best.

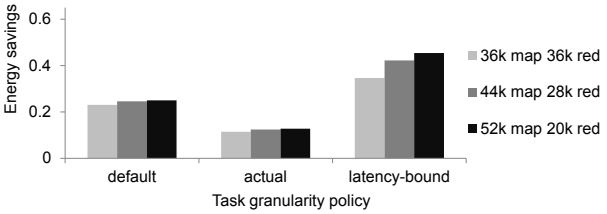


Figure 12. Energy savings for different values of `mapreduceratio`. Increasing the number of map slots increases energy savings for all `taskcalc` algorithms, with the improvement for *latency-bound* being the greatest. Note that `totalsize` is set to 72000 slots, `batchlen` is set to 24 hours, and `interruptible` is set to 24 hours.

the week shown in Figure 11(a). Aggregated over the entire trace, the actual policy turns out to have more dangling tasks overall, diminishing energy savings.

In contrast, task slot occupancy over time for the latency-bound policy eliminates all dangling tasks of long durations (Figure 11(b)). This results in high cluster utilization during batches, as well as clean batch completion, allowing the cluster to be transitioned into a low-power state at the end of a batch. There is still room for improvement in Figure 11(b): the active reduce slots are still far from reaching maximum task slot capacity. This suggests that even if we keep the total number of task slots constant, we can harness more energy savings by changing some reduce slots to map slots.

5.4 Map to Reduce Slot Ratio

The evaluation thus far illustrates that reduce slots are utilized less than map slots. Changing `mapreduceratio` (i.e., increasing the number of map slots and decreasing the number of reduce slots while keeping cluster size constant) should allow map tasks in each batch to complete faster without affecting reduce tasks completion rates. Figure 12 shows that doing so leads to energy efficiency improvements, especially for the latency-bound algorithm.

Viewing the task slot occupancy over time reveals that this intuition about the map-to-reduce-slot ratio is correct. Figure 13(a) compares batch zone slot occupancy for two different ratios using the default algorithm. With a larger number of map slots, the periods of maximum map slot occupancy are shorter, but there are still dangling reduce tasks. The same ratio using the latency-bound algorithm

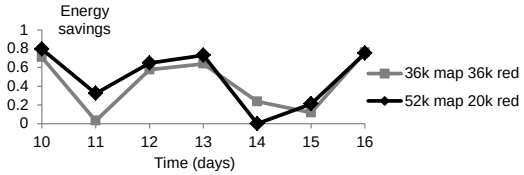


Figure 14. Energy savings per day for the latency-bound policy comparison in Figure 13(b). Daily energy savings range from 0 to 80%. Neither static policy achieves best energy savings for all days.

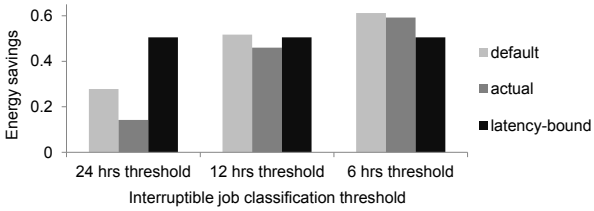


Figure 15. Energy savings for different values of `interruptible`. Lowering the threshold leads to increased energy savings for actual and default algorithms. Note that `mapreduceratio` is set to 13:5 and `batchlen` is set to 24 hours. Note that for actual and default algorithms, having a low `interruptible` causes the queue for waiting interrupted jobs to grow without limit; the latency-bound policy is preferred despite seemingly lower energy savings (Section 5.5).

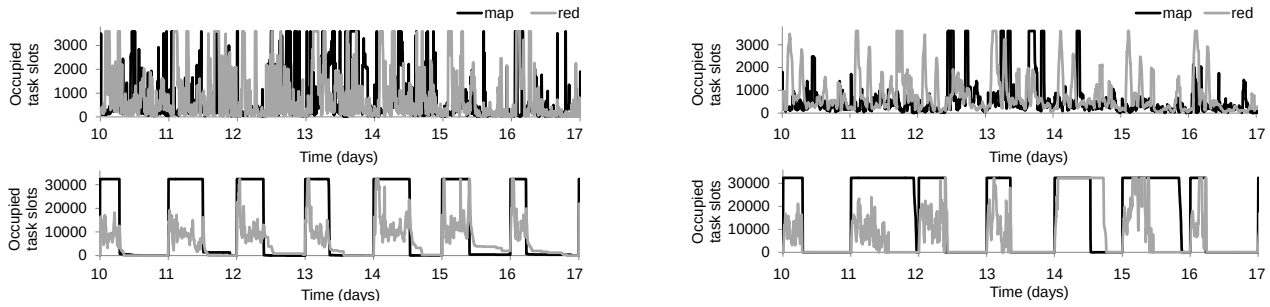
avoids these dangling reduce tasks, as shown in Figure 13(b), achieving higher energy savings.

Nevertheless, the latency-bound algorithm still has room for improvement. During the fifth and sixth days in Figure 13(b), the batches are in fact limited by available reduce slots. Figure 14 shows that neither static policy for map versus task ratios achieve the best savings for all days. A dynamically adjustable ratio of map and reduce slots is best. A dynamic ratio can ensure that every batch is optimally executed, bottlenecked on neither map slots nor reduce slots.

5.5 Interruptible Threshold

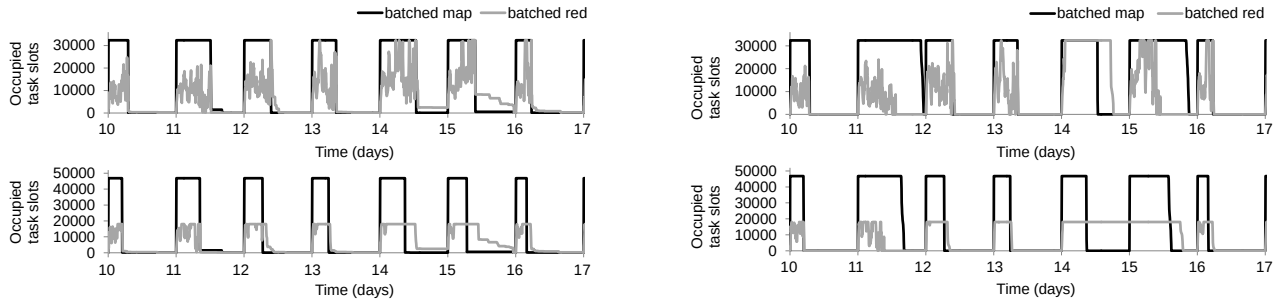
The last dimension to evaluate is `interruptible`, the task duration threshold that determines when a job is classified as interruptible. In the evaluation so far, `interruptible` has been set to 24 hours. Decreasing this threshold should cause more jobs to be classified as interruptible, and fewer jobs as batch. A lower interruptible threshold allows faster batch completions and potentially more capacity for the interactive zone, at the cost of higher average job latency, as more jobs are spread over multiple batches.

Figure 15 shows the energy saving improvements from lowering `interruptible`. (The latency-bound algorithm, by design, does not result in any interruptible jobs, unless the `interruptible` is set to less than an hour, so the energy savings for the latency-bound algorithm are unaffected.) Actual and default algorithms show considerable energy savings improvements, at the cost of longer latency for some jobs. It would be interesting to see how many cluster users and administrators are willing to make such trades.



(a) *Actual* (Facebook) (b) *Latency-bound*

Figure 11. Slot occupancy over time in the interactive zone (top graph) and batch zone (bottom graph). Showing one week's behavior. Note that `batchlen` is set to 24 hours, `mapreduceratio` is set to 1:1, and `interruptible` is set to 24 hours.



(a) *Default* (b) *Latency-bound*

Figure 13. Batch zone slot occupancy over time using a `mapreduceratio` of 1:1 for the top graph, and a `mapreduceratio` of 13:5 for the bottom graph. Showing one week's behavior. Note that `batchlen` is set to 24 hours and `interruptible` is set to 24 hours.

Lowering `interruptible` too much would cause the queue of waiting interruptible jobs to build without bound. Consider the ideal-case upper bound on possible energy savings. The Facebook workload has a historical average of 21029 active map tasks and 7745 active reduce tasks. A cluster of 72000 task slots can service 72000 concurrent tasks at maximum. Thus, the best case energy savings is $1 - (21029 + 7745)/72000 = 0.60$. As we lower `interruptible`, any energy "savings" above this ideal actually represents the wait queue building up.

The best policy combination we examined achieves energy savings of 0.55 fraction of the baseline, as shown Figure 15, with `taskcalc` set to default and `interruptible` set to 6 hours. This corresponds to 92% of this ideal case.

5.6 Overhead

The energy savings come at the cost of increased job latency. Figure 16 quantifies the latency increase by looking at normalized job durations for each job type. BEEMR achieves minimal latency overhead for interactive jobs, and some overhead for other job types. This delayed execution overhead buys us energy savings for non-interactive jobs.

For interactive jobs, more than 60% of jobs have ratio of 1.0, approximately 40% of jobs have ratio less than 1.0, and a few outliers have ratio slightly above 1.0. This indicates that a dedicated interactive zone can lead to either unaffected job latency, or even improved job latency from having dedicated resources. The small number of jobs with ratio above 1.0 is caused by peaks in interactive job arrivals. This suggests that it would be desirable to increase the capacity of the interactive zone during workload peaks.

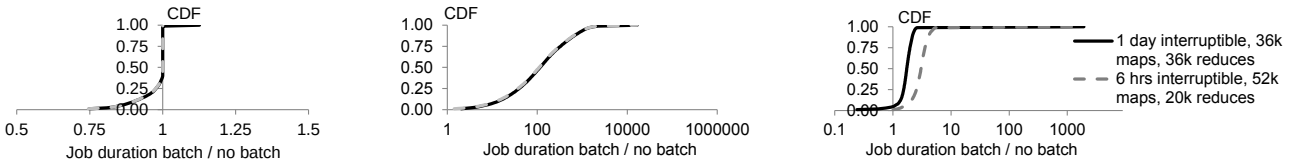
For batched jobs, the overhead spans a large range. This is caused by the long batch interval, and is acceptable as a matter of policy. A job that arrives just after the beginning of one batch would have delay of at least one batch interval, leading to large latency. Conversely, a job that arrives just before a batch starts will have almost no delay. This is the same delayed execution behavior as policies in which users specify, say, a daily deadline.

For interruptible jobs, the overhead is also small for most jobs. This is surprising because interruptible jobs can potentially execute over multiple batches. The result indicates that interruptible jobs are truly long running jobs. Executing them over multiple batches imposes a modest overhead.

5.7 Sensitivity

The evaluation thus far has set a `totalsize` of 72000 task slots and discovered the best parameter values based on this setting. A cluster size of 72000 forms a conservative baseline for energy consumption. Using BEEMR on larger clusters yields more energy savings, as shown in Figure 17.

BEEMR extracts most, but not all, of the ideal energy savings. The discrepancy arises from long tasks that hold up batch completion (Section 5.2) and transient imbalance be-



(a) Interactive jobs (b) Batch jobs (c) Interruptible jobs

Figure 16. Latency ratio by job type between BEEMR with `totalsize` set to 72000, `taskcalc` set to default, and (1) `batchlen` set to 24 hours, `mapreduceratio` set to 1:1, or (2) `batchlen` set to 6 hours, `mapreduceratio` set to 13:5; versus the baseline with no batching. A ratio of 1.0 indicates no overhead. Some interactive jobs see improved performance (ratio < 1) due to dedicated resources. Some batch jobs have very long delays, the same behavior as delayed execution under deadline-based policies. Interruptible jobs have less overhead than batch jobs, indicating that those are truly long running jobs. The delayed execution in non-interactive jobs buys us energy savings.

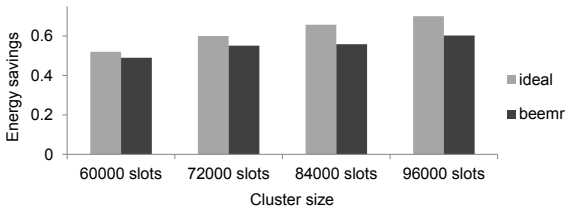


Figure 17. Ideal and observed energy savings for different cluster sizes. Both increase as cluster size increases. Note that `batchlen` is set to 24 hours, `taskcalc` is set to default, `mapreduceratio` is set to 13:5, and `interruptible` is set to 6 hours.

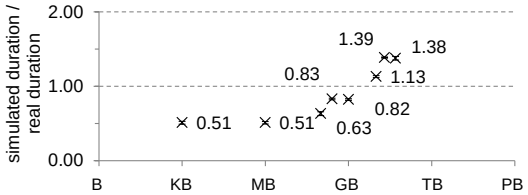


Figure 18. Simulator validation for stand-alone jobs. Showing the ratio between simulated job duration and average job duration from 20 repeated measurements on a real cluster. The ratio is bounded for both large and small jobs and is very close to 1.0 for sort jobs of size 100s of MB to 10s of GB.

tween map and reduce slots (Section 5.4). If the fraction of time that each batch runs at maximum slot occupancy is already small, then the effects of long tasks and map/reduce slot imbalance are amplified. Thus, as cluster size increases, the gap between BEEMR energy savings and the ideal also increases. One way to narrow the gap would be to extend the batch interval length, thus amortizing the overhead of long tasks holding up batch completion and transient map/reduce slot imbalance. In the extreme case, BEEMR can achieve arbitrarily close to ideal energy savings by running the historical workload in one single batch.

5.8 Validation

Empirical validation of the simulator provides guidance on how simulation results translate to real clusters. The BEEMR simulator explicitly trades simulation scale and speed for accuracy, making it even more important to quantify the simulation error.

We validate the BEEMR simulator using an Amazon EC2 cluster of 200 "m1.large" instances [1]. We ran three experiments: (1) a series of stand-alone sort jobs, (2) replay several day-long Facebook workloads using the methodology in [12], which reproduces arrival patterns and data sizes using synthetic MapReduce jobs running on synthetic data, (3) replay the same workloads in day-long batches. For Experiments 1 and 2, we compare the job durations from these experiments to those obtained by a simulator configured with the same number of task slots and the same policies regarding task granularity. For Experiment 3, we compare the energy savings predicted by the simulator to that from the EC2 cluster. These experiments represent an essential validation step before deployment on the actual front-line Facebook cluster running live data and production code.

Figure 18 shows the results from stand-alone sort jobs. This ratio is bounded on both ends and is very close to 1.0 for sort jobs of size 100s of MB to 10s of GB. The simulator underestimates the run time (the ratio is less than 1.0) for small sort sizes. There, the overhead of starting and terminating a job dominates; this overhead is ignored by the simulator. The simulator overestimates the run time (the ratio is greater than 1.0) for large sort sizes. For those jobs, there is non-negligible overlap between map and reduce tasks; this overlap is not simulated. The simulation error is bounded for both very large and very small jobs.

Also, there is low variance between different runs of the same job, with 95% confidence intervals from 20 repeated measurements being barely visible in Figure 18. Thus, pathologically long caused by task failures or speculative/abandoned executions are infrequent; not simulating these events causes little error.

Figure 19 shows the results of replaying one day's worth of jobs, using three different day-long workloads. The ratio is again bounded, and close to 0.75 for the majority of jobs. This is because most jobs in the workload have data sizes in the MB to GB range (Figure 1). As explained previously,

Figure 19. Simulator validation for three day-long workloads, without batching. Showing the ratio between simulated and real job duration. This ratio is bounded on both ends and is very close to 0.75 for the vast majority of jobs.

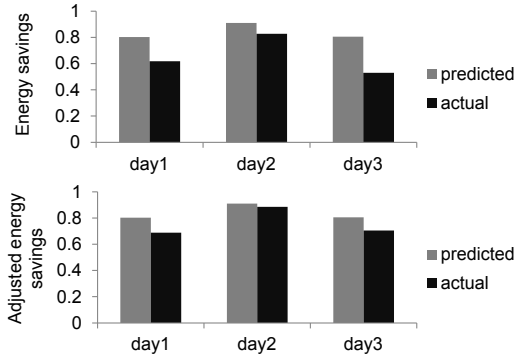


Figure 20. Simulator validation for three different day-long workloads, with `batchlen` set to 24 hours. Showing the predicted versus actual energy savings (top graph, average 22% simulation error), and the predicted versus actual energy savings after adjusting for the slot occupancy capacity on the real-life cluster (bottom graph, average 13% simulation error).

job startup and termination overhead lead to the simulator to underestimate the duration of these jobs.

Figure 20 shows the validation results from batching the three day-long workloads. The simulation error varies greatly between three different days. The average error is 22% of the simulated energy savings (top graph in Figure 20). We identify two additional sources of simulator error: (1) The BEEMR simulator assumes that all available task slots are occupied during the batches. However, on the EC2 cluster, the task slot occupancy averages from 50% to 75% of capacity, a discrepancy again due to task start and termination overhead — the scheduler simply cannot keep all task slots occupied. Adjusting the simulator by using a lower cluster size than the real cluster yields the bottom graph in Figure 20, with the error decreased to 13% of the simulated energy savings. (2) The BEEMR simulator assumes that task times remain the same regardless of whether the workload is executed as jobs arrive, or executed in batch. Observations from the EC2 cluster reveals that during batches, the higher real-life cluster utilization leads to complex interference between jobs, with contention for disk, network, and other resources. This leads to longer task times when a workload executes in batch, and forms another kind of simulation error that is very hard to model.

Overall, these validation results mean that the simulated energy savings of 50-60% (Section 5.5) would likely translate to 40-50% on a real cluster.

6. Discussion

The results in Section 5 raise many interesting questions. Some additional issues await further discussion below.

6.1 Power Cycles versus Reliability

Transitioning machines to low-power states is one way to achieve power proportionality for MIA workloads while more power proportional hardware is being developed. Large scale adoption of this technique has been limited by worries that power cycling increases failure rates.

There have been few published, large-scale studies that attribute increased failure rates to power cycling. The authors in [41] observed a correlation between the two, but point out that correlation could come simply from failed systems needing more reboots to restore. To identify a causal relationship would require a more rigorous methodology, comparing mirror systems servicing the same workload, with the only difference being the frequency of power cycles.

One such comparison experiment ran for 18 months on 100s of machines, and found that power cycling has no effect on failure rates [39]. Larger scale comparisons have been stymied by the small amount of predicted energy savings, and uncertainty about how those energy savings translate to real systems. BEEMR gives empirically validated energy savings of 40-50%. This represents more rigorous data to justify further exploring the thus far unverified relationship between power cycles and failure rates.

6.2 MIA Generality Beyond Facebook

MIA workloads beyond Facebook lend themselves to a BEEMR-like approach. We analyzed four additional Hadoop workloads from e-commerce, telecommunications, media, and retail companies. These traces come from production clusters of up to 700 machines, and cover 4 cluster-months of behavior. The following gives a glimpse of the data. We are seeking approval to release these additional workloads.

One observation that motivated BEEMR is that most jobs access small files that make up a small fraction of stored bytes (Figure 4). This access pattern allows a small interactive zone to service its many jobs. Figure 21shows that such access patterns exist for all workloads. For FB-2010, input paths of $<$ 10GB account for 88% of jobs and 1% of stored bytes. For workloads A and D, the same threshold respectively accounts for 87% and 87% of jobs, and 4% and 2% of stored bytes. For workloads B and C, input paths of $<$ 1TB accounts for 86% and 91% of jobs, as well as 12% and 17% of stored bytes.

Another source of energy savings comes from the high peak-to-average ratio in workload arrival patterns (Figure 2). The cluster has to be provisioned for the peak, which makes

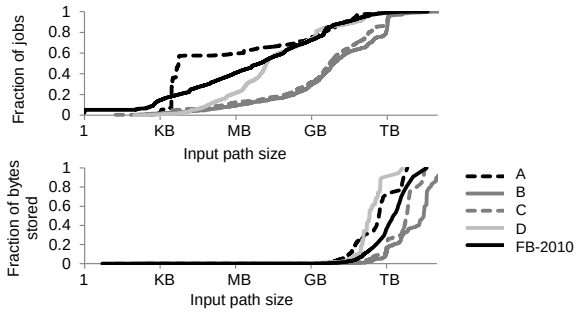


Figure 21. Access patterns vs. input path size. Showing cummulative fraction of jobs with input paths of a certain size (top) and cummulative fraction of all stored bytes from input paths of a certain size (bottom). Contains data from Figure 4 for the FB-2010 workload, and four additional workloads from e-commerce, telecommunications, media, and retail companies.

it important to achieve energy proportionality either in hardware or by workload managers such as BEEMR. For the five workloads (Facebook and workloads A through D), the peak-to-average ratios are: 8.9, 30.5, 23.9, 14.5, and 5.9. BEEMR potentially extracts higher energy savings from workloads with higher peak-to-average arrival ratios, though the exact energy savings and the tradeoff between policy parameters is workload specific. These additional workloads give us confidence that the BEEMR architecture can generalize beyond the Facebook workload.

6.3 Methodology Reflections

Evaluating the energy efficiency of large scale distributed systems presents significant methodological challenges. This paper strikes a balance between scale and accuracy. Future work could improve on our techniques.

Simulation vs. replay. The inherent difference between MIA and other workloads suggest that the best energy efficiency mechanisms would be highly workload dependent. Even for MIA workloads, the behavior varies between use cases and over time (Figures 14 and 21). Thus, only evaluation over long durations can reveal the true historical savings (Figure 14). Days or even weeks-long experiments are unrealistic, especially to explore multiple design options at large scale. Hence, we are compelled to use simulations.

Choice of simulator. We considered using Mumak [37] and MRPerf [53]. Mumak requires logs generated by the Rumen tracing tool [44], which is not yet in universal use and not used at Facebook. MRPerf generates a simulation event per control message and per packet, which limits simulation scale and speed. Neither simulator has been verified at the multi-job workload level. Thus, we developed the BEEMR simulator, which intentionally trades simulation detail and accuracy to gain scale and speed. We also verify the simulator at the workload level (Section 5.8).

Choice of power model. One accurate way to measure system power is by a power meter attached at the machine wall socket [11]. This method does not scale to clusters of 1000s of machines. The alternative is to use empirically verified power models, which are yet to be satisfactorily developed for MapReduce. The translation between SPECpower [49] measurements and MapReduce remains unknown, as it is between MapReduce workload semantics and detailed CPU, memory, disk, and network activity. We chose an on-off power model, i.e., machines have "max" power when on and "zero" power when off. This simple model allow us to scale the experiments in size and in time.

Towards improved methodology. The deliberate tradeoffs we had to make reflect the nascent performance understanding and modeling of large scale systems such as MapReduce. We encourage the research community to seek to overcome the methodology limitations of this study.

6.4 Future Work for MapReduce in General

Designing and evaluating BEEMR has revealed several opportunities for future improvements to MapReduce.

1. The BEEMR policy space is large. It would be desirable to automatically detect good values for the policy parameters in Table 3.
2. The ability to interrupt and resume jobs is desirable. This feature is proposed under Next Generation Hadoop for fast resume from failures [38]. Energy efficiency would be another motivation for this feature.
3. A well-tuned `taskcalc` algorithm can significantly affect various performance metrics (Section 5.3). However, choosing the correct number of tasks to assign to a job remains an unexplored area. Given recent advances in predicting MapReduce execution time [20, 36], we expect a dedicated effort would discover many improvements.
4. The chatty HDFS/Hadoop messaging protocols limits the dynamic power of machines to a narrow range. There is an opportunity to re-think such protocols for distributed systems to improve power proportionality.
5. The disjoint interactive and batch zones can be further segregated into disjoint interactive and batch clusters. Segregated versus combined cluster operations need to balance a variety of policy, logistical, economic, and engineering concerns. More systematic understanding of energy costs helps inform the discussion.
6. The gap between ideal and BEEMR energy savings increases with cluster size (Section 5.7). It is worth exploring whether more fine-grained power management schemes would close the gap and allow operators to provision for peak while conserving energy costs.

7. Closing Thoughts

BEEMR is able to cut the energy consumption of a cluster almost *in half* (after adjusting for empirically quantified simulation error) without harming the response time of latency-sensitive jobs or relying on storage replication, while allowing jobs to retain the full storage capacity and compute bandwidth of the cluster. BEEMR achieves such

results because its design was guided by a thorough analysis of a real-world, large-scale instance of the targeted workload. We dubbed this widespread yet under-studied workload MIA. The key insight from our analysis of MIA workloads is that although MIA clusters host huge volumes of data, the interactive jobs operate on just a small fraction of the data, and thus can be served by a small pool of dedicated machines; the less time-sensitive jobs can run in a batch fashion on the rest of the cluster. We are making available the sanitized Facebook MIA workload traces (`https://github.com/SWIMProjectUCB/SWIM/wiki`) to ensure that ongoing efforts to design large scale MapReduce systems can build on the insights derived in this paper.

References

[1] Amazon Web Services. Amazon Elastic Computing Cloud. http://aws.amazon.com/ec2/.

[2] G. Ananthanarayanan et al. Scarlett: coping with skewed content popularity in mapreduce clusters. In *Eurosys 2011*.

[3] R. H. Arpaci et al. The interaction of parallel and sequential workloads on a network of workstations. In *SIGMETRICS 1995*.

[4] I. Ashok and J. Zahorjan. Scheduling a mixed interactive and batch workload on a parallel, shared memory supercomputer. In *Supercomputing 1992*.

[5] L. A. Barroso. Warehouse-scale computing: Entering the teenage decade. In *ISCA 2011*.

[6] C. Belady. In the data center, power and cooling costs more than the IT equipment it supports. *Electronics Cooling Magazine*, Feb. 2007.

[7] R. Bianchini and R. Rajamony. Power and energy management for server systems. *Computer*, Nov. 2004.

[8] D. Borthakur. Facebook has the world's largest Hadoop cluster! `http://hadoopblog.blogspot.com/2010/05/facebook-has-worlds-largest-hadoop.html`.

[9] D. Borthakur et al. Apache Hadoop goes realtime at Facebook. In *SIGMOD 2011*.

[10] L. Breslau et al. Web Caching and Zipf-like Distributions: Evidence and Implications. In *INFOCOM 1999*.

[11] Y. Chen, L. Keys, and R. H. Katz. Towards Energy Efficient MapReduce. Technical Report UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.

[12] Y. Chen et al. The Case for Evaluating MapReduce Performance Using Workload Suites. In *MASCOTS 2011*.

[13] Y. Chen et al. Statistical Workloads for Energy Efficient MapReduce. Technical Report UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[14] J. Corbet. LWN.net 2009 Kernel Summit coverage: How Google uses Linux. 2009.

[15] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. *Comm. of the ACM*, 51(1):107–113, January 2008.

[16] Q. Deng et al. Memscale: active low-power modes for main memory. In *ASPLOS 2011*.

[17] EMC and IDC iView. Digital Universe. `http://www.emc.com/leadership/programs/digital-universe.htm`.

[18] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *Micro, IEEE*, 28(3):42 –53, May-June 2008.

[19] X. Fan, W.-D. Weber, and L. A. Barroso. Power provisioning for a warehouse-sized computer. In *ISCA 2007*.

[20] A. Ganapathi et al. Statistics-driven workload modeling for the cloud. In *ICDEW 2010*.

[21] J. Gray et al. Quickly generating billion-record synthetic databases. In *SIGMOD 1994*.

[22] Gridmix. HADOOP-HOME/mapred/src/benchmarks/gridmix2 in Hadoop 0.20.2 onwards.

[23] Hadoop World 2011. Hadoop World 2011 Speakers. `http://www.hadoopworld.com/speakers/`.

[24] J. Hamilton. Overall Data Center Costs. `http://perspectives.mvdirona.com/2010/09/18/OverallDataCenterCosts.aspx`, 2010.

[25] Hewlett-Packard Corp., Intel Corp., Microsoft Corp., Phoenix Technologies Ltd., Toshiba Corp. Advanced Configuration and Power Interface 5.0. `http://www.acpi.info/`.

[26] M. Isard et al. Quincy: fair scheduling for distributed computing clusters. In *SOSP 2009*.

[27] R. T. Kaushik et al. Evaluation and Analysis of GreenHDFS: A Self-Adaptive, Energy-Conserving Variant of the Hadoop Distributed File System. In *IEEE CloudCom 2010*.

[28] W. Lang and J. Patel. Energy management for mapreduce clusters. In *VLDB 2010*.

[29] J. Leverich and C. Kozyrakis. On the Energy (In)efficiency of Hadoop Clusters. In *HotPower 2009*.

[30] P. Lieberman. White paper: Wake on lan technology, June 2006.

[31] B. Liu et al. A study of networks simulation efficiency: Fluid simulation vs. packet-level simulation. In *Infocom 2001*.

[32] D. Meisner et al. Power management of online data-intensive services. In *ISCA 2011*.

[33] D. Meisner et al. Powernap: eliminating server idle power. In *ASPLOS 2009*.

[34] S. Melnik et al. Dremel: interactive analysis of web-scale datasets. In *VLDB 2010*.

[35] A. K. Mishra et al. Towards characterizing cloud backend workloads: insights from Google compute clusters. *SIGMETRICS Perform. Eval. Rev.*, 37:34–41, March 2010.

[36] K. Morton et al. ParaTimer: a progress indicator for MapReduce DAGs. In *SIGMOD 2010*.

[37] Mumak. Mumak: Map-Reduce Simulator. `https://issues.apache.org/jira/browse/MAPREDUCE-728`.

[38] A. Murthy. Next Generation Hadoop Map-Reduce. Apache Hadoop Summit 2011.

[39] D. Patterson. Energy-Efficient Computing: the State of the Art. Microsoft Research Faculty Summit 2009.

[40] Personal email. Communication regarding release of Google production cluster data.

[41] E. Pinheiro et al. Failure trends in a large disk drive population. In *FAST 2007*.

[42] G. F. Riley, T. M. Jaafar, and R. M. Fujimoto. Integrated fluid and packet network simulations. In *MASCOTS 2002*.

[43] S. Rivoire et al. Joulesort: a balanced energy-efficiency benchmark. In *SIGMOD 2007*.

[44] Rumen: a tool to extract job characterization data from job tracker logs. `https://issues.apache.org/jira/browse/MAPREDUCE-751`.

[45] A. Ryan. Next-Generation Hadoop Operations. Bay Area Hadoop User Group, February 2010.

[46] J. H. Saltzer. A simple linear model of demand paging performance. *Commun. ACM*, 17:181–186, April 1974.

[47] K. Shvachko. HDFS Scalability: the limits to growth. *Login*, 35(2):6–16, April 2010.

[48] D. C. Snowdon et al. Accurate on-line prediction of processor and memory energy usage under voltage scaling. In *EMSOFT 2007*.

[49] SPEC. SPECpower 2008. `http://www.spec.org/power_ssj2008/`.

[50] The Green Grid. The Green Grid Data Center Power Efficiency Metrics: PUE and DCiE, 2007.

[51] A. Thusoo et al. Data warehousing and analytics infrastructure at Facebook. In *SIGMOD 2010*.

[52] U.S. Environmental Protection Agency. Report to Congress on Server and Data Center Energy Efficiency, Public Law 109-431, 2007.

[53] G. Wang et al. A simulation approach to evaluating design decisions in MapReduce setups. In *MASCOTS 2009*.

[54] M. Zaharia et al. Delay scheduling: a simple technique for achieving locality and fairness in cluster scheduling. In *EuroSys 2010*.