Why let resources idle? Aggressive Cloning of Jobs with Dolly

Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, Ion Stoica

University of California, Berkeley
{ganesha,alig,shenker,istoica}@cs.berkeley.edu

Abstract

Despite prior research on outlier mitigation, our analysis of jobs from the Facebook cluster shows that outliers still occur, especially in small jobs. Small jobs are particularly sensitive to long-running outlier tasks because of their interactive nature. Outlier mitigation strategies rely on comparing different tasks of the same job and launching speculative copies for the slower tasks. However, small jobs execute all their tasks simultaneously, thereby not providing sufficient time to observe and compare tasks. Building on the observation that clusters are underutilized, we take speculation to its logical extreme—*run full clones of jobs to mitigate the effect of outliers*. The heavy-tail distribution of job sizes implies that we can impact most jobs without using much resources. Trace-driven simulations show that average completion time of all the small jobs improves by 47% using cloning, at the cost of just 3% extra resources.

1 Introduction

Cloud computing has become a significant technological breakthrough. An increasing number of organizations use datacenters to run a mixed variety of computations, ranging from long-running batch jobs to interactive short queries that operators launch on the fly.

The importance of these datacenter computations has led to much effort being spent on optimizing their performance. The prevalence of outlier tasks was early identified as a common source of performance problem [1]. Initial research suggested the use of speculative execution to mitigate such outliers. These methods were later improved by LATE [2] and Mantri [3], which provide more intelligent outlier mitigation based on speculative execution of tasks. Similar techniques have also been used to deal with outliers in other settings [4, 5].

Despite this research on outlier mitigation, our analyses of traces from a $3{,}500$ node Facebook cluster, that applies the LATE technique, shows that outliers are still common, especially in small jobs. The small jobs, on average, have outlier tasks that are 12 times slower than that job's median task, which significantly delays completion of jobs. Our simulations show that the outlier numbers for Mantri are similar for small jobs.

Small jobs are particularly sensitive to outliers because they execute in a single wave of simultaneously running tasks. Therefore even a single task being an outlier slows down the entire job. The single-waved property also limits the efficacy of traditional outlier mitigation strategies that rely on comparing different tasks of the same job. Any meaningful comparison requires waiting to obtain statistically significant samples of task performance, which single-waved small jobs cannot afford.

In this work, we focus on improving the completion time of these small jobs, which are often interactive queries, where the response time is important to the human operator awaiting its response. The idea we explore in this paper is to take speculative execution to its logical extreme and *run full clones of jobs to reduce job completion times*. Two trends make this approach viable.

First, most jobs are small and consume few resources. Our analysis shows that job sizes have a power-law distribution, with the absolute majority of the jobs being small, while the absolute majority of the cluster resources are spent on a small number of large jobs. Thus, the aggregate resources consumed by small jobs is moderate. Running clones of small jobs has the potential to impact most jobs, without using much resources.

Second, most clusters are highly underutilized. Several of the clusters that we analyzed have a very low average utilization. In particular, CPU and memory utilization in these clusters has a median less than 20%. In fact, cluster utilization exceeds the 50% mark only 8% of the time. There is thus room for running extra clones of jobs.

A key question is whether running job clones will negatively impact energy efficiency. Despite research on powering down machines for energy efficiency, we note that most clusters today *do not* shut off machines to save energy. Thus, machines are on most of the

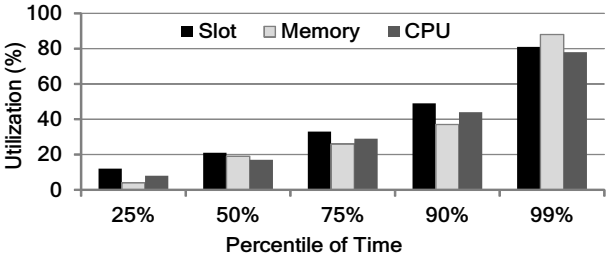
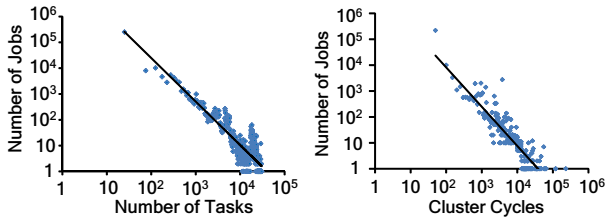


Figure 1: **Distribution of cluster utilization values.**



(a) Number of tasks (b) Cluster cycles

Figure 2: **Power-law distribution of jobs in the number of tasks and cluster cycles consumed. Power-law exponents are** 1.9 **and** 1.7 **when fitted with least squares regression.**

time, and the majority of the energy cost is already incurred as the underlying hardware has limited power-proportionality [6]. Further, the energy costs of operating datacenters is largely proportional to the peak-utilization and not just the actual utilization [7]. Our position is, therefore, to use these idling resources for useful work.

Our job cloning strategy, called Dolly,[1] uses a probabilistic model to dynamically account for the probability of occurrence of an outlier in individual machines. Based on this model, it launches several clones of tasks to reach a target probability of outlier-free executions. Dolly is configured to never increase the utilization of the cluster beyond a prescribed threshold, and will immediately abort clones in case utilization crosses that threshold.

Our evaluation shows that Dolly, when used for cloning only small jobs (the smallest 82% of jobs), improves their completion time by 47% and 39%, compared to LATE and Mantri, respectively, at the cost of increasing utilization by merely 3%. This improves the overall average completion time of all jobs by 40% and 33% in the two cases. A more aggressive use of Dolly would be to use it for cloning all jobs. This improves the overall average completion time by 44% at the expense of increasing utilization of the cluster by a factor of 2.2×, well within reach of today's underutilized clusters.

2 Making the Case for Cloning

Our work rests on two complementary characteristics of data-intensive clusters—low utilization of resources and occurrence of outliers in small jobs. We elaborate on this using traces from Facebook's Hadoop [8] cluster. These traces represent 375K jobs over a month's duration.

2.1 Low Utilization

We analyze utilization across slots, CPU and memory. A slot, allotted to tasks, typically refers to a pre-assigned number of compute cores and memory. Slot utilization is the fraction of time the slots in the cluster were running tasks. CPU utilization is the fraction of time the cores in the cluster were utilized for computation. Memory utilization measures the memory occupancy by tasks.

[1] Dolly was the first mammal (sheep) to be cloned.

Figures 1 lists the distribution of utilizations. Clearly, resources remain idle. The median slot, CPU and memory utilizations is only 21%, 19% and 17%, respectively. In other words nearly four-fifths of the cluster remains idle for half the time. In fact, utilizations exceed 50% for only 8% of the time. Such trends of low utilization are consistent with prior observations (*e.g.,* [9, 10]).

This raises the question of why are these clusters so heavily over-provisioned. The answer is peak utilization. The peak utilization of the Facebook cluster exceeds 99%. Clusters are provisioned for their peak utilizations, even if that occurs for only a small period, to provide consistent SLA guarantees on job performance. Unfortunately, this leaves the cluster idling at other times.

Workload shaping can spread the load on the cluster and lower peak utilization (as in [11]). However, implementing it in practice, especially when there are user-driven jobs, is complicated. Regardless, efforts at reducing the peak-to-average ratio is beyond the scope of this paper. We believe that cluster utilizations will continue to be low and systems should appropriately make use of the available idling resources for their benefit.

2.2 Outliers in Jobs

Before quantifying outlier tasks in jobs, we first describe the mix of jobs that run in these clusters.

A common property of workloads in clusters is their heavy-tail distribution—many small jobs dominate by count and a few large jobs consume most of the resources [12].[2] Small jobs are interactive computations submitted by users, who then wait for their completion and hence are sensitive to delays in completion times. In the Facebook cluster, 82% of jobs have less than 10 tasks in them. However, these jobs consume only 6% of the aggregate resources of the cluster. Indeed, as Figure 2 shows, the number of tasks in jobs, and cluster cycles consumed by jobs, follow a power-law distribution.

[2] The terms "small" and "large" can equivalently refer to the input size, cluster cycles, or the number of tasks of a job. Cluster cycles of a job is defined as the sum of the duration of its tasks.

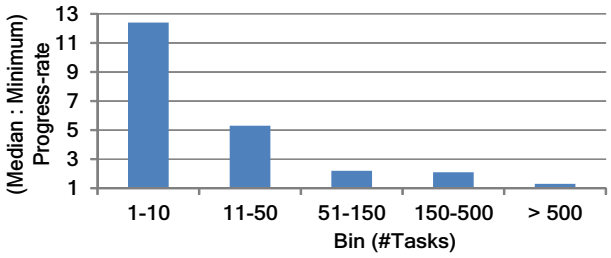


Figure 3: **Ratio of median to minimum progress rates for tasks of a job. Small jobs are more impacted by outliers.**

Outlier tasks significantly hamper jobs in the Facebook cluster even though the LATE mitigation strategy [2] is in operation. LATE speculatively executes tasks that are estimated to finish last. We measure the magnitude of outliers by comparing *progress rates* of tasks within a job. The progress rate of a task is defined as its input size divided by its duration. A measure of how significantly a job is affected by outliers is given by the ratio of median to minimum progress rates among its tasks.[3] A high value indicates the presence of outliers. Figure 3 shows the variation of this ratio across jobs of different sizes. Small jobs have this ratio to be over 12.

Outliers particularly hurt small jobs. Small jobs run all their tasks simultaneously in one *wave*. Therefore, a delay in completion of even a single task holds up the entire job's completion. This is unlike jobs that run over multiple waves of tasks. For such multi-waved jobs, impact of outliers can be better absorbed across the waves.

Before explaining the idea of cloning, we would like to point out the lack of any correlation between outliers and either high utilization or a choice of machines. Let the median to maximum ratio of progress rates in a time window be the median of the ratios across all the jobs that were active in that window. The Pearson correlation coefficient between the ratios and utilization is -0.12, implying little correlation. In fact the median utilization of the cluster when the median to maximum ratio is ≥ 3 is 22.7%, close to the overall median utilization. Also, outliers are not caused by a small set of persistently problematic machines. This is not surprising as the Facebook cluster already blacklists machines with faulty disks and other hardware troubles using periodic diagnostics

2.3 Cloning: Idea and Potential

The observations in §2.1 and §2.2—low utilization and prevalence of outliers, especially in but not limited to small jobs—points to an opportunity to *aggressively utilize the idle resources to clone jobs thereby probabilistically mitigating the effect of outliers.*

[3] We use only successfully completed tasks for our analysis and discard those tasks that failed or were preempted (*e.g.,* as in [13, 14]).

The traditional approach for mitigating outliers is speculation [1, 2, 3]. Speculative copies are spawned for those tasks that are deemed to be slower. It then becomes a race between the speculative and original copies. Without getting in to the details of the algorithms that trigger the speculative copies, we point out that these algorithms decide on which tasks to speculate based on observed performance of other tasks of the same job. The time taken to generate statistically significant observations limits agility of the speculation algorithm. Further, outlier tasks are not necessarily slow during their entire duration. It is often the case that they become slow when they are well into their processing. Spawning a speculative copy at that point might be late. This is evidenced by the outliers affecting jobs in the Facebook cluster in spite of the LATE speculation strategy [2] being used.

In this paper, we argue for speculation of jobs just as they are launched. We call such speculative copies that are not dependent on any performance observations as *clones*. If there are idle resources, clones are created. The result is chosen from the clone that finishes first.

The idea of cloning rests on the following two observations. First, an outlier does not occur due to any inherent computational property but due to systemic eccentricities. Therefore it is less likely to occur if another copy is spawned elsewhere. Second, the probability of an outlier decreases exponentially with the number of clones.

Potential Improvement: If all outliers are hypothetically ironed out, the average completion time of jobs improves by 46% and small jobs ($\leq$ 10 tasks) improve by 49%. Encouraged by this potential, we try a strawman scheme—clone all small jobs ($\leq$ 10 tasks) by simply running multiple instances of the whole job. The number of clones is sufficient to limit the chances of an outlier in the job to 5% (we defer the exact details to §3.1). Even this simple strategy improves the average completion time of small jobs by 35% ($\sim 70\%$ of ideal).

2.4 Energy Efficiency

The idea of using idle resources aggressively is counter to proposals that make clusters energy-efficient [15, 16]. Such proposals advocate powering down machines during idle periods. However, to the best of our knowledge, most clusters have not adopted these proposals. There are many reasons behind this lack of adoption.

Recent research reports that the operating energy expense of a datacenter is largely influenced by the peak consumption, even if that is only for a short period. In fact, for modern datacenters, the cost of peak consumption constitutes as high as 40% of the total expense. Therefore, the energy-proportional proposals address only the remaining part of the operational expense. This has proved insufficient for cluster operators.

An engineering consideration, but important noneth-

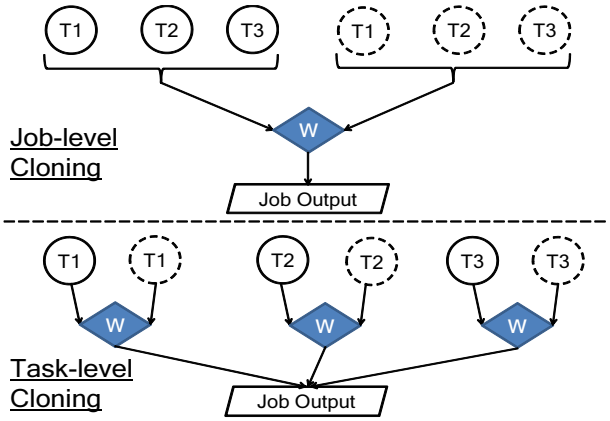
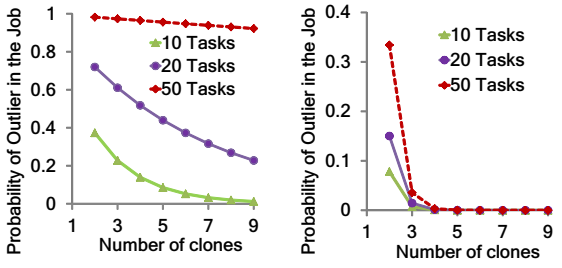


Figure 4: **Illustration of job-level and task-level cloning for a job with 3 tasks with and 2 clones. The rhombus with "W" picks the winner between the clones, *i.e.,* the earliest.**



(a) Job-level Cloning (b) Task-level Cloning

Figure 5: **Probability of a job having an outlier for varying number of clones. We use sample job sizes of 10, 20 and 50 tasks. Task-level cloning requires fewer replicas.**

less, behind the lack of adoption of power-cycling of machines is the resulting unpredictability of the application software as well as the hardware. Application software, that run continuously on machines, produce unpredictable behavior when frequently restarted. Hence the preference to keep machines on continuously. Commodity hardware degrades in reliability when continuously power-cycled. Conversations with datacenter operators confirm these concerns as the reason behind their unenthusiastic response to power-cycling in clusters.

Finally, powering off machines results in lower return on the hardware investment. Clusters typically procure new hardware every few years. With such recurring capital costs, it is arguably beneficial for them to explore ways to utilize their sunk costs instead of powering them down, the direction that we are pursuing with Dolly.

3 Design Issues in Dolly

We now discuss about the design details of Dolly.

3.1 Cloning Granularity

The first question that arises is the granularity of cloning. As broached in §2.3, one option is to clone at the granularity of jobs. For every job submitted to the cluster, multiple clones are spawned. Results are returned from the earliest clone of the job. Such job-level cloning is appealing due to its simplicity. The alternative to this is task-level cloning. Every *task* is cloned and a winner is picked between the clones of every task. The output of that task is passed to tasks downstream in the job. Therefore, unlike job-level cloning, task-level cloning requires intricate changes to the execution of the job. Figure 4 illustrates the difference between the two cloning techniques: job-level cloning is successful only if all the tasks of a clone do not become outliers.

For the same number of clones, task-level cloning provides better guarantees for eliminating outliers compared to job-level cloning. Let p be the probability of a task becoming an outlier. For a job with n tasks and c clones, the probability that it has an outlier is $(1-(1-p)^n)^c$ with job-level cloning, and $1-(1-p^c)^n$ with task-level cloning. Figure 5 compares how these probabilities decrease as we increase the number of clones. Task-level cloning gains more per clone and the probability of the job having an outlier drops off considerably faster. In fact, for 96% of jobs in the cluster, task-level cloning results in $\leq 5\%$ chance of outliers with just three replicas.

Despite low utilization (§2.1) in clusters, task-level cloning's resource efficiency is desirable. It reduces contention among clones for network links, disk bandwidth and memory locality [12, 17] (see §4). Job-level cloning, on the other hand, is simpler to implement and transparent to the internal details of the frameworks, albeit at the expense of efficiency. A complete comparison of the two cloning techniques is part of future work.

3.2 Probability of Outlier

Estimating the probability of occurrence of an outlier is crucial for Dolly because it dictates the number of clones to spawn. An inflated value wastes unnecessary clones on a job denying resources to other jobs while an underestimation does not effectively mitigate outliers.

We observe that the cluster-wide probability of a task being an outlier is roughly constant over time. Further, there are no select set of machines that consistently cause outliers. Nonetheless, we observe considerable variations in the probability per machine over time.

The probability of occurrence of an outlier on a machine (p_m) is defined as the ratio of the number of outliers on that machine to the total number of tasks that it executed. We define an outlier as a task whose progress rate is less than $2\times$ the median progress rate of tasks in its job.[4] We calculate the coefficient-of-variation $\left(\frac{stdev}{mean}\right)$

[4]We are actively considering more sophisticated definitions to categorize outliers including clustering techniques.

procedure CLONE_TASK-LEVEL(Job with n tasks, p)
  $c_o = \log\left(1 - \epsilon^{(1/t)}\right) / \log p$
  M: $\langle$(name, p_m)$\rangle$ ▷ Machines sorted by descending p_m
  for Task t **do**
    $p_t \leftarrow p_o$ ▷ Outlier probability of original copy
    $i \leftarrow 0$ ▷ Index for machine list
    while $p_t \times$ M[i].$p_m > p^{c_o}$ **and** $(U+1) < \tau$ **do**
      Clone t on M[i].name
      $p_t \leftarrow p_t \times$ M[i].p_m
      $i \leftarrow i + 1$
      $U \leftarrow U + 1$

Pseudocode 1: **Task-level cloning for a job with n tasks on a cluster with overall probability of outlier as p.**

of p_m across machines in every hourly window. The median of these coefficients over the entire duration of the Facebook trace is 2.12, indicating a high variation in p_m between machines. Coupled with the observations that the cluster-wide probability is constant across time and no set of machines are persistently problematic, this indicates the need to periodically learn the probability of occurrence of an outlier on each machine.

Learning based on an average of recent and historical values yields encouraging accuracy. We define the probability at the end of time-window t to be $p_{m,t+1} = (p_{m,t} + p_m[t])/2$ where $p_m[t]$ is the probability during the t^{th} time-window. This predicts with an accuracy of 56%. Using improved learning models is underway.

3.3 Cloning Algorithm

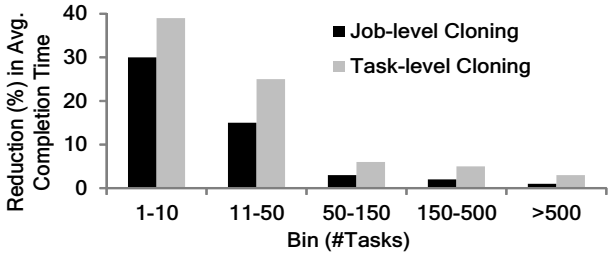
Cloning is conditional on the cluster's current utilization. Cloning happens if the expected utilization after spawning clones is less than a ceiling of τ, a value that can be set based on operational considerations. Current slot utilization (U) is measured as the number of running tasks.

Given this setting, the three parameters of interest for cloning jobs are the cluster-wide probability of an outlier (p), probability of outliers per machine (p_m), and the acceptable probability of a job having an outlier (ϵ).

We first describe job-level cloning. We use the cluster-wide probability of an outlier, p. For a job with n tasks, the number of clones required, c, can be derived to be at least $\frac{\log \epsilon}{\log(1-(1-p)^n)}$. The number of clones actually spawned is limited by the available slots within the utilization threshold: $\min\left(c, \left\lfloor \frac{\tau - U + c \cdot n}{n} \right\rfloor\right)$.

Task-level cloning, on the other hand, considers the outlier probabilities per machine. It first calculates an indicative number of clones (c_o) using the cluster-wide outlier probability p. For cloning individual tasks, it looks at the set of machines in the cluster in descending order of their outlier probabilities. It picks the smallest set of machines (*i.e.,* number of clones) whose combined probability of eliminating an outlier is greater than when all

(a) Baseline: LATE



(b) Baseline: Mantri

Figure 6: **Reduction in average completion time using** Dolly **compared to the LATE and Mantri speculation strategies.**

the machines had a probability of p and the task ran c_o clones. Pseudocode 1 describes the algorithm in detail.

Finally, cloning of jobs should be transparent to fairness guarantees. An incoming job which is entitled to get its fair share, could request the slot of a running task. Dolly prioritizes the incoming job's task over all the copies of the running task but one; only the original copy is given equal priority as the incoming task. Therefore, an incoming task can immediately claim its fair share by killing clones in progress. Such killing of clones can also be used during emergency to bring down utilization.

Evaluation: We evaluate Dolly's performance using a trace-driven simulator replaying Hadoop logs from the Facebook cluster. We used an ϵ of 0.05 and τ of 70%. Facebook's cluster deploys LATE and so that is our baseline. Overall average completion time of jobs reduce by 44% and 31% with task-level and job-level cloning, respectively. Figure 6a breaks down the improvement by job sizes. Note that this difference is despite job-level cloning consuming $5.8\times$ more resources. While the difference between the two cloning strategies is significant for small jobs (≤ 10 tasks), task-level cloning's resource efficiency assumes greater importance for larger jobs.

We also compare Dolly to Mantri's [3] outlier mitigation techniques. Mantri speculates tasks based on a comparison between the estimated remaining time for a running task and the expected duration of a speculative copy, calculated using samples from completed tasks of the job. Figure 6b shows the benefits of cloning when the baseline is Mantri. Average completion time improves

by 35% and 26% for the two cloning strategies.

It is noteworthy that if only the small jobs were cloned the overall average completion time still reduces by 40% and 33% compared to LATE and Mantri, at the cost of increasing utilization by merely 3.3% (*i.e.,* from 21% to 24.3%). This is due to the power-law distribution of job sizes (Figure 2), a major reason behind Dolly's efficacy.

Dolly's gains are limited for large jobs where LATE and Mantri's thoughtful speculation is already effective. We envision Dolly's *proactive* cloning to co-exist with such *reactive* outlier mitigation techniques.

4 Future Directions

As we move forward with the development of Dolly, addressing I/O contention among clones will be important.

Input Data: Clones are less likely to contend on disk bandwidth because input data is typically replicated three times by file systems, and small jobs require fewer clones. However, clones will contend for memory bandwidth and memory local slots when inputs are cached in memory [12, 17], which usually is not replicated. Replicating files in memory can be challenging because memory capacity is already three orders of magnitude lower than disks. Our analysis of Facebook traces shows that 92% of the smallest active jobs can fit their data in memory because of the power-law distribution [18]. Since the desired number of clones is low for small jobs, we can replicate the cached inputs of only the small jobs at the expense of evicting some input blocks of large jobs.

Intermediate Data: Cloning poses challenges when it comes to handling the intermediate data transferred between the map and reduce phases. Cloned reduce tasks need to read their data from the map tasks that produced the data. Because of the all-to-all traffic pattern of shuffles, a reduce task often needs to fetch data from every map task's output. Thus, frameworks does not attempt to provide locality for reduce tasks. Further, intermediate data is not replicated to avoid resource and time overheads. This means that Dolly should take care such that, (i) cloned reduce tasks do not overwhelm the machine they are fetching the map output from, and (ii) the extra data necessary to feed the clones does not congest the network. A thorough analysis of dealing with these problems is our immediate future plan, but we next explain the problem of contention for intermediate map outputs.

Ideally, we would like to assign an exclusive copy of map output to every reduce clone. However, when some of the map clones are outliers, there are fewer copies of the map outputs. This presents us with a tricky situation in assigning intermediate map outputs to reduce clones. Care has to taken to balance the risk of waiting for a map clone to complete against the overhead of letting reduce clones contend for the available copies of map outputs.

Finally, we plan to perform a full-fledged implementation of Dolly inside data-intensive frameworks (*e.g.,* Hadoop), and deployment to evaluate its benefits.

Acknowledgments

We thank Facebook for access to Hadoop job traces from their production cluster. For feedback on improving the draft, we thank Arka Bhattacharya, Mosharaf Chowdhury, Aurojit Panda, and David Zats. This research was supported by the sponsors of the AMP Lab at Berkeley: SAP, Amazon Web Services, Cloudera, Huawei, IBM, Intel, Microsoft, NEC, NetApp and VMWare, and by DARPA (contract #FA8650-11-C-7136).

References

[1] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *USENIX OSDI*, 2004.

[2] M. Zaharia, A. Konwinski, A. D. Joseph, R. Katz, I. Stoica. Improving MapReduce Performance in Heterogeneous Environments. In *USENIX OSDI*, 2008.

[3] G. Ananthanarayanan, S. Kandula, A. Greenberg, I. Stoica, E. Harris, and B. Saha. Reining in the Outliers in Map-Reduce Clusters using Mantri. In *USENIX OSDI*, 2010.

[4] C. Wilson, H. Ballani, T. Karagiannis, A. Rowstron. Better Never than Late: Meeting Deadlines in Datacenter Networks. In *SIGCOMM*, 2011.

[5] S. Melnik, A. Gubarev, J. J. Long, G. Romer, S. Shivakumar, M. Tolton, T. Vassilakis. Dremel: Interactive Analysis of Web-Scale Datasets. In *VLDB*, 2010.

[6] D. Meisner, C. M. Sadler, L. A. Barroso, W-D Weber, and T F. Wenisch. Power management of online data-intensive services. In *ISCA*, 2011.

[7] S. Govindan, A. Sivasubramaniam, B. Urgaonkar. Benets and Limitations of Tapping into Stored Energy For Datacenters. In *ISCA*, 2011.

[8] Hadoop. `http://hadoop.apache.org`.

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

[10] Y. Chen, S. Alspaugh, D. Borthakur, R. Katz. Energy Efficiency for Large-Scale MapReduce Workloads with Significant Interactive Analysis. In *ACM EuroSys*, 2012.

[11] L. Ganesh, J. Liu, S. Nath, G. Reeves, F. Zhao. Unleash stranded power in data centers with rackpacker. In *WEED*, 2009.

[12] G. Ananthanarayanan, A. Ghodsi, A. Wang, D. Borthakur, S. Kandula, S. Shenker, I. Stoica. PACMan: Coordinated Memory Caching for Parallel Jobs. In *USENIX NSDI*, 2012.

[13] M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg. Quincy: Fair Scheduling for Distributed Computing Clusters. In *ACM SOSP*, 2009.

[14] G. Ananthanarayanan, S. Agarwal, S. Kandula, A. Greenberg, I. Stoica, D. Harlan, E. Harris. Scarlett: Coping with Skewed Popularity Content in MapReduce Clusters. In *EuroSys*, 2011.

[15] A. Krioukov, P. Mohan, S. Alspaugh, L. Keys, D. E. Culler and R. H. Katz. NapSAC: The design and implementation of a power proportional web cluster. In *ACM GreenNets*, 2010.

[16] Y. Chen, L. Keys, R. Katz. Towards Energy Efficient Hadoop. In *Hadoop Summit*, 2009.

[17] 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.

[18] G. Ananthanarayanan, A. Ghodsi, S. Shenker, I. Stoica. Disk Locality Considered Irrelevant. In *USENIX HotOS*, 2011.