Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay

Yanpei Chen
Archana Sulochana Ganapathi
Rean Griffith
Randy H. Katz

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2010-81
http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-81.html

May 15, 2010

Towards Understanding Cloud Performance Tradeoffs Using Statistical Workload Analysis and Replay

Yanpei Chen, Archana Ganapathi, Rean Griffith, Randy H. Katz
University of California, Berkeley

Abstract

Cloud computing has given rise to a variety of distributed applications that rely on the ability to harness commodity resources for large scale computations. The inherent performance variability in these applications' workload coupled with the system's heterogeneity render ineffective heuristics-based design decisions such as system configuration, application partitioning and placement, and job scheduling. Furthermore, the cloud operator's objective to maximize utilization conflicts with cloud application developers' goals of minimizing latency, necessitating systematic approaches to tradeoff these optimization angles. One important cloud application that highlights these tradeoffs is MapReduce.

In this paper, we demonstrate a systematic approach to reasoning about cloud performance tradeoffs using a tool we developed called Statistical Workload Analysis and Replay for MapReduce (SWARM). We use SWARM to generate realistic workloads to examine latency-utilization tradeoffs in MapReduce. SWARM enables us to infer that batched and multi-tenant execution effectively balance the tradeoff between latency and cluster utilization, a key insight for cloud operators.

1 Introduction

Cloud computing has made it easier for application developers to harness resources for large-scale distributed computations. While application developers desire faster response times, both public and private cloud operators seek to trade performance for higher resource utilization, and thus lower operating costs.

MapReduce has emerged as a popular computation paradigm for processing large quantities of data in both public and private clouds. Many Internet enterprises, i.e. private clouds, rely on MapReduce for their core business analytics and data mining applications. We use it to highlight the conflicting goals between cloud operators and application developers. There is significant benefit to Cloud operators understanding performance trade-offs in MapReduce-style computations, so they can customize design and policy choices to trade per-job latency against whole workload resource utilization.

The diversity of MapReduce usage scenarios makes it difficult to develop a single performance benchmark. We believe a statistics-based systematic approach can assist cloud operators in understanding complex tradeoffs for achieving predictable performance.

Design decisions must be informed by multiple factors including, but not limited to, latency, resource utilization and performance variability. These factors often conflict with one another. A reasonable tradeoff must satisfy both the application developer's performance goals and the operator's cost constraints.

Our work is motivated by the need for more detailed investigation of the multi-dimensional performance tradeoffs in MapReduce and the need for workload synthesis and replay tools for facilitating these investigations. To that end, the contributions presented in this paper are as follows:

1. Analysis of production MapReduce traces from two large scale private clouds. We collect production traces from two Internet services. We analyze the empirical workload characteristics to identify potential sources of variability that affect performance. We bootstrap our workload synthesis and replay tool with the trace data to produce more representative workloads than existing pseudo-benchmarks such as sort and gridmix.

2. Workload synthesis and replay using statistical techniques. We present SWARM, a tool for statistics-based workload synthesis and replay. SWARM distills the statistical distributions for workload characteristics of interest from production traces, and samples these empirical distributions to generate a job stream that mimics workload characteristics. This two-step approach can be generalized to other applications by merely changing the traces used for bootstrapping SWARM. Using SWARM,

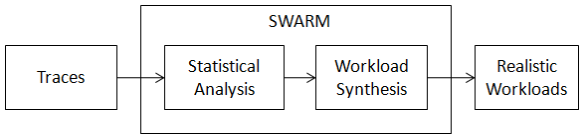


Figure 1: SWARM process flow showing workload synthesis (top) and replay (bottom).

we generate synthetic yet representative workloads that can drive our exploration of performance tradeoffs.

3. Von-Neumann-Morgenstern (VNM) utility function formulation for evaluating cloud performance tradeoffs. We use SWARM-generated workloads in a public cloud environment and VNM utility functions to investigate performance tradeoffs between latency and utilization. VNM utility is an established formulation we borrowed from economic game theory. In particular, the VNM formulation invites cloud operators to re-examine their definitions of "good" performance.

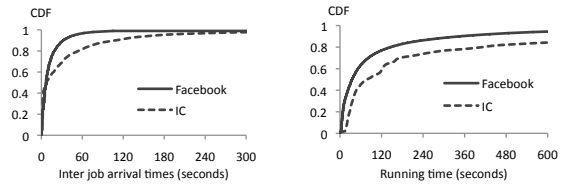
4. Insights for cloud operators. Our key insights are: 1. MapReduce performance variance is large, yet the statistical distribution of performance is static. 2. The tradeoff between utilization and latency can be made using batch execution and multi-tenant workloads operating modes. 3. Data intensive workloads benefit from running on smaller nodes with lower I/O performance, provided that the bottleneck is degrees of parallelism and not the datapath itself.

Figure 1 summarizes our workload synthesis process. We consume traces, produce statistical distributions for workload characteristics, and then produce representative workload through sampling the statistical distributions. Then we replay the synthesized workloads on MapReduce clusters to explore the performance space.

The remainder of this paper is organized as follows. Section 2 extracts insights from two production MapReduce workloads to bootstrap statistical models used by SWARM. Section 3 presents details of SWARM's workload synthesis and replay mechanisms and describes the generated synthetic workload and experimental testbed we use for our work. In Section 4, we evaluate tradeoffs between latency and utilization using VNM utility functions. We demonstrate the effectiveness of this formulation for understanding design tradeoffs in the public cloud. Lastly, we compare related work, in Section 5, and conclude with insights on our approach for exploring distributed system performance tradeoffs.

2 Lessons from Two Production Traces

We obtained production MapReduce traces from two Internet enterprises. One dataset – from *Facebook (FB)* – represents 6 months of activity on a 600 node dedicated Hadoop cluster at Facebook. The second dataset



(a) **(b)**

Figure 2: Comparison of Facebook and Internet Company temporal metrics.

– from *Internet Company (IC)*[1] – contains 2 weeks of activity on a 40 node Hadoop on Demand [8] cluster. Both datasets depict multi-user environments running the Hadoop implementation of MapReduce on clusters dedicated to MapReduce workloads.

A brief overview of MapReduce is helpful at this point. At its core, MapReduce has two user-defined functions. The Map function takes in a key-value pair, and generates a set of intermediate key-value pairs. The Reduce function takes all intermediate pairs associated with a particular key, and emits a final set of key-value pairs. The runtime system schedules parallel execution, handles machine failures, shuffles the intermediate data, and reads input from/writes output to an underlying distributed file system (DFS). For more implementation details, we refer the reader to [13].

We can describe a MapReduce job by several dimensions - the inter-job arrival time, the input/shuffle/output data sizes, the computation performed by the map and reduce functions, the number of map and reduce tasks, the running time, data locality, and others. Whereas the traces we have do not allow a comprehensive characterization along all these dimensions, we describe the empirical phenomena along the dimensions that are present. We focus on statistical constructs to capture workloads characteristics, which form the basis of workload synthesis and replay in Section 3. Thus, while the traces do not reflect all use cases of MapReduce, they offer sufficient snapshot for synthesis and replay.

Inter-job arrival times. Figure 2(a) shows the cumulative distribution function (CDF) of inter-job arrival times for both our datasets. Most inter-job arrival intervals last tens of seconds. In comparison, the gridmix pseudo-benchmark launches jobs in quick succession with negligible inter-job arrival time, capturing only one extreme of the statistical distribution. The two statistical distributions differ both in shape and in average value. Thus, a single inter-arrival distribution cannot represent both workloads well.

Running time. Figure 2(b) represents the distribution of running time of jobs in our datasets. Both distributions have similar shape, although the distribution for the

[1] actual name anonymized

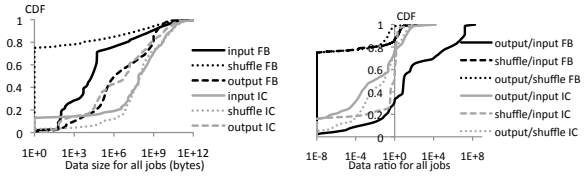


Figure 3: Comparison of data sizes and data ratios for Facebook and Internet Company jobs.

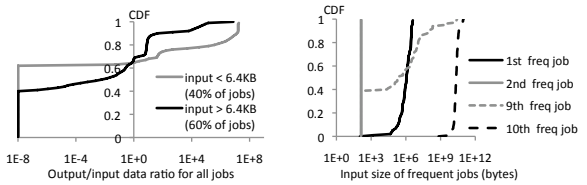


Figure 4: Facebook trace analysis - (a) CDF of output/input data ratio grouped by 40th percentile in input size (b) CDF of input sizes for several frequent jobs.

IC trace appears to be horizontally shifted to a higher mean. Running time is an essential performance metric. However, one can argue that running time is a joint function of the workload and the workload execution environment, influenced by factors such as scheduling, data placement, etc. If the goal is to capture characteristics intrinsic only to workload, then any synthesis and replay mechanisms should not try to reproduce running time.

Data sizes. Figure 3(a) shows the data sizes per job at various stages of the MapReduce pipeline. The shapes of the curves do not follow any well known parametric distributions. Furthermore, data sizes range from KB to TB for both datasets. The gridmix benchmark only captures an extreme end of the workload by focusing solely on GB and TB data sizes. When realistically replaying workloads, we must capture the entire data size range.

Data ratios. Figure 3(b) shows the distribution of data ratios for shuffle-input, output-shuffle and output-input data sizes. Again, for the two traces, the statistical distributions differ both in shape and in average value. The graph indicates that many jobs have output-input ratios far from 1. Data expansion jobs, such as loading files from file name inputs, will yield output-input ratios far above 1. Conversely, data aggregation jobs, such as computing data aggregates, will result in output-input ratios far below 1. These two types of jobs routinely appear in production environments with big data processing needs. Thus, workload synthesis should accurately reproduce the mix of data expansion and aggregation jobs.

Data ratios depend on data sizes. Figures 4(a) and 4(b) respectively show the CDF of output/input data ratio for big and small input jobs, and the input data size CDF for very frequent jobs in the Facebook trace. These

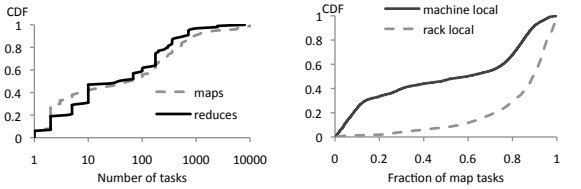


Figure 5: Internet Company trace analysis - (a) CDF of map and reduce task counts per job. (b) CDF of the fraction of machine and rack local map tasks per job.

distributions are very different from the CDF of all output/input data ratios in Figure 3(b) and the CDF of all input sizes in Figure 3(a), indicating a three-way dependency between the job, the distribution of its data sizes, and the distribution of its data ratios. A good workload generation mechanism should capture this dependency.

Map and reduce tasks counts. Figure 5(a) shows the distribution of the number of map and reduce tasks per job. We have this data for only the IC trace. The tasks per job range across several orders of magnitude. The distribution shows some bimodal behavior, with the transition region between 10 tasks and 100 tasks per job.

Data locality. Figure 5(b) shows the distribution of the fraction of data local map tasks. Again, we have this data for only the IC trace. Most jobs contain a high fraction of rack local or machine local map tasks. However, the fraction of strictly machine local tasks has a wider distribution. The machine local distribution also shows some bimodal behavior, with the interval from 0.3 to 0.7 being the transition region. Similar to running time, data locality is arguably a function of workload execution, rather than an intrinsic property of the workload.

Data skew. The traces do not presently contain information about data skew. However, we are confident that such data exists, and we are working with our industry partners to obtain this data.

Map and reduce functions computations The traces do not contain descriptions of the map and reduce functions for each job. Arguably, such information would enable the reverse engineering of key business processes and should thus remain proprietary. For in-house trace analysis, we believe it would be insightful to distill the common types of map and reduce computations within a production workload, since this would enable a more detailed characterization of the workload.

CPU, memory, disk, and network utilization. The traces do not presently contain information on cluster resource utilization. Although it would be insightful to look at this data, we believe that this data is not an inherent property of the workload, since workload execution characteristics such as placement and network topology affect these metrics.

3 SWARM and Experimental Testbed

Production traces inform the synthesis of representative workloads for multi-dimensional performance comparisons. In this section, we describe SWARM - Statistical Workload Analysis and Replay in MapReduce. We explain the systematic process SWARM uses to produce synthetic workload. We also describe the Amazon EC2 public cloud environment we use for experiments.

3.1 Workload Synthesis

SWARM first samples the CDF of inter-job arrival times and the PDF of job names. Then, for each sampled job name, it samples the input data size, shuffle-input and output-shuffle data ratios for that particular job name. Thus we construct a workload consisting of vectors of jobs described by [*inter-job arrival time*, *job name*, *input size*, *shuffle-input ratio*, and *output-shuffle ratio*]. We repeat sampling until we reach the desired number of jobs or the desired time interval length for the workload.

To ensure fidelity of our synthetic workload to the original traces, we must at least mimic inter-job arrival times and the input, shuffle, and output data sizes. Our sampling also reproduces data ratios and their per-jobname dependency. We can include other characteristics intrinsic to the workload, such as data skew and the computational task being done, but currently omit these characteristics as such information is absent in our traces.

The inability to characterize computational behavior is unfortunate. For data intensive rather than compute intensive. [11] suggests that the precise map and reduce computations performed have less impact on performance than quantity of data used for data intensive workload. SWARM currently targets data intensive workloads. However, for in-house experiments, our sampling framework can extend to capture the distribution of computational tasks.

Faithfully reproducing trace behavior involves a fundamental trade-off. If the workload generator exactly mimics the trace's characteristics, then the replay likely reproduces quirks and defects of the system that generated the trace. In particular, any characteristics open to design changes should not be included in the generated workload. Therefore, we exclude map and reduce task counts, running times, data locality, and resource utilization metrics from our generated workload as they are a function of the workload execution environment.

A final but crucial detail is that we scale input data size by the ratio between the size of the original trace generation cluster and the size of the cluster where workload is replayed, preserving the amount of data per node. The scaling is necessary because one cannot expect, the production input data set to fit on clusters that are orders of magnitude smaller in the number of nodes. Some large scale behavior is inevitably lost by scaling down the data. However, one can argue that faithfully measuring large scale behavior must use a production cluster with production data sets. Thus, scaled down experiments represent a necessary prerequisite to justify using production resources for performance evaluation.

3.2 Workload Replay

The primary objective for our framework is to replay realistic workload. To this end, the SWARM workload executor takes our synthetic workload vectors and produces a sequence of MapReduce jobs with varying characteristics. At a high level, our workload generator consists of one or more shell scripts that launch jobs with specified data sizes and data ratios, and sleeps between successive jobs to account for inter-arrival times. Specifically, our workload replay tool comprises of three components.

First, SWARM uses *RandomWriter* to populate input data, creating multiple fixed size files, one for each reduce task of a particular job. We populate the input data once per workload, accounting for the jobs' required maximum input data size. Jobs in a workload select a random subset of these files as input based on the desired input data size. We set each file to be 64MB, the same granularity as the default HDFS block size. We also validated that there is negligible performance overhead from concurrent jobs reading from the same HDFS input.

Next, we implement a *RatioMapReduce* job that reproduces input-shuffle and output-shuffle ratios specified in the workload vector. We use a simple probabilistic identity filter, constructed by modifying the RandomWriter example included with the Hadoop source code.

Lastly, we delete each MapReduce job's output to prevent storage capacity overload. Upon completion of each job in the workload, run the *HDFS remove* command as a background process to remove data generated by that job. We experimentally ensured that this mechanism imposes no performance overhead.

3.3 Generated Workloads

We generated two workloads – the Facebook workload and a bimodal workload. The Facebook workload, as the name suggests, comes from sampling the Facebook traces. The bimodal workload demonstrates the flexibility of our workload generation tools, allowing us to construct arbitrary workloads with more challenging in data size, job frequency, and variance between jobs. The bimodal workload is composed of two jobs types. Large jobs comprise 0.5% of all jobs and have 500 MB input per node. Small jobs comprise the other 99.5% of jobs in the workload and have 50 MB of input data per node

producing a higher average data size than the Facebook workload. Both the large and small jobs have shuffle-input and output-shuffle ratio of 1.0. The inter-arrival time is exponentially distributed, with an average of 15 seconds - again more intensive than the Facebook workload. We believe that bimodal behavior could potentially arise in some production environments. Figures 5 and 5 already display some bimodal behavior, albeit in secondary workload characteristics.

We do not generate synthetic workloads based on IC traces as they represent a shorter time interval (2 weeks) and smaller cluster (40 nodes) compared to the Facebook traces (6 months of data from a 600 node cluster).

3.4 Replay Environment

To evaluate design considerations at scale, we conduct experiments in Amazon's Elastic Cloud Computing (EC2). EC2 allows users to select from a set of virtual machine instance types. Most of our experiments are on m1.large instances, with 7.5 GB memory and 4 EC2 Compute Units, priced at $0.38 per hour for the Northern California availability zone. We also performed experiments on m1.small instances to quantify cost-performance tradeoffs. The m1.small instances have 1.7 GB memory and 1 EC2 Compute Unit, priced at $0.095 for Northern California. [2]

We use the Hadoop implementation of MapReduce [2], distribution 0.18.2 running on Java version 6.0, update 11. We avoid newer versions to ensure data comparability with previous experiments and results in [12, 15]. We used default configuration parameters for Hadoop.

4 Performance Results and Analysis

We investigate several aspects of MapReduce workload performance using the Facebook and bimodal workloads described previously. To quantify performance tradeoffs, we first run each workload in isolation on EC2 nodes. Then, we run both workloads simultaneously on shared EC2 nodes to understand multi-tenant superposition effects. The key lessons we learned include:

1. The distribution of performance variation is fixed. Thus, cumulative distribution functions (CDFs) effectively represent performance variation.

2. We can decrease performance variation and increase resource utilization through batched execution, with a cost in increased per-job latency.

3. We can tradeoff latency and resource utilization using suitable utility functions for each. The Von Neumann - Morgenstern utility is a helpful formulation.

[2]Each Compute Unit is equivalent to the CPU capacity of a 1.0-1.2 GHz 2007 Opteron or Xeon processor.

4. We use the VNM utility formulation to assess tradeoffs in several other dimensions, including different machines types and cluster sizes. Counter-intuitively, data intensive workloads benefit from running on smaller nodes with lower I/O performance, provided that the bottleneck is degrees of parallelism and not the datapath itself.

5. Multi-tenant execution involves the same utilization-latency tradeoff as batched execution, with the VNM utility again finding a good operating point.

Our goal in this section is to discuss in detail the experiments, results, and analysis that led to the above lessons. Following this discussion, we will summarize the implications for MapReduce operators in the next section.

4.1 Static Performance Distribution

To quantify MapReduce performance variation at scale, we run the Facebook and bimodal workloads independently on Hadoop clusters of 100 m1.large EC2 instances. We run each workload in batch intervals of 5 minutes, 15 minutes, 1 hour, and 1 day, with two repeated runs for each batch setting. The measurements from these experiments reveal that MapReduce workload performance varies significantly, but the distribution of variation remains fixed. Thus we can effectively characterize performance variation using CDFs.

MapReduce workload performance varies significantly. Figure 6 shows the per-batch running time for different batch intervals. For the same batch interval, the bimodal workload shows higher variance. For our experiments, performance variance is a joint function of the workload and the batch size.

Performance distribution remains fixed. When we have a large number of batches, although per-batch performance varies significantly, the performance distribution remains fixed. Figure 7 shows the CDF of running times for two different batch sizes for our two workloads. There is a negligible difference between the two lines representing two repeated runs in Figure 7. We verified that the same holds for other batch sizes and workloads.

Capturing performance distribution using CDFs. The CDF is a good way to capture performance variation because it has two properties - it remains fixed from run to run, and it describes the entire performance distribution. There are alternatives to capturing performance distributions, such as using the 90th, 99th, 99.9th percentiles to represent statistical summaries. Two distributions could have, the same 99th percentile yet completely different behavior below the 99th percentile. In contrast, two distributions with the same CDF would be the same according to any statistical summary. For these reasons, we believe that CDFs would provide a more detailed and complete specification of service level objectives com-

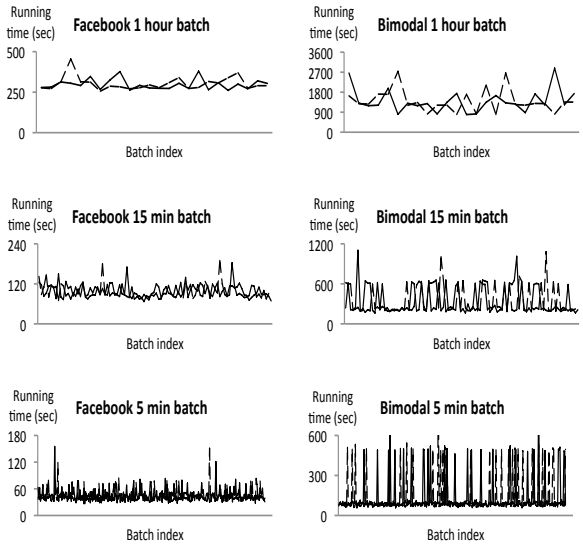


Figure 6: Batch running times between repeated measurements for different batch interval sizes. The two lines show the two repeated measurements.

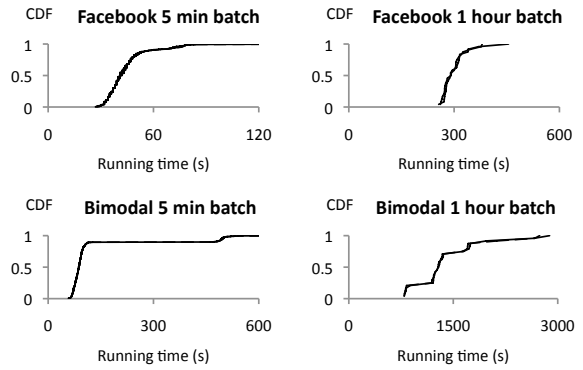


Figure 7: Batch running time CDFs.

pared to the 90th, 99th, 99.9th percentiles.

4.2 Benefits of Batching

We have already seen in Figure 6 that larger batch intervals correspond with smaller variation. Additional analysis shows that there is in fact a fundamental causal relationship between the two metrics. Also, larger batch intervals correspond with higher resource utilization. These observations lead to our second lesson - for a small increase in latency, we can decrease performance variation and increase resource utilization through batching.

Normalized performance CDFs. To compare the running time CDFs for different batch interval sizes on the same numerical scale, we must normalize the running time by batch interval size. We introduce the term *active fraction* to refer to the running time normalized as follows - the ratio between the running time of a particular batch and the batch interval represents the fraction of the batch interval during which MapReduce was actively doing work. Figure 8 shows the active fraction CDFs for the two workloads at different batch intervals. Figure 8 offers another perspective on the correlation between large batch intervals and small variance. The CDFs of the larger batch intervals are "narrower" in the horizontal direction and thus have smaller variation.

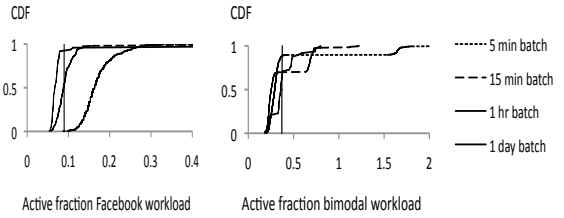


Figure 8: Active fraction CDFs.

Larger batches result in higher utilization. Per our definition, active fraction represents the fraction of the batch interval during which MapReduce was actively doing work. Thus, a lower active fraction represents higher time efficiency and higher resource utilization. Figure 8 shows that larger batch intervals correspond to lower active fractions. This relationship is straightforward to understand. At the beginning of each batch, the cluster would have high utilization. However, as jobs complete, any straggling would cause the cluster to be active at a low utilization. In the limit of infinitely small batches, i.e., no batching, the active fraction would include any time that the cluster had a job to run, even if a large cluster is utilized at a low level to accomplish a small task. This tradeeoff between utilization/throughput and latency is well known. One can use our measurement method to quantify the tradeoff for computing paradigms other than MapReduce.

Larger batch intervals decrease variance. There is in fact a strong, causal relationship between batch interval size and variance in per batch running time. This relationship can be explained using the Central Limit Theorem, which states the following. If we sample many times from any statistical distribution with finite mean μ and variance σ^2, then as the number of samples n gets large, the sum of samples converge to a normal (Gaussian) distribution with mean μ and variance σ^2/n. In other words, we decrease the variance in the sum simply by summing over more things. The connection to MapReduce workloads is as follows. We can write the per-batch running time as

$$RunningTime = T + \varepsilon$$

where T represents the total per-batch running time, and ε is the per-batch inherent variance induced by Hadoop and the underlying physical infrastructure. As batch interval sizes increase, there are more jobs, translating to more samples of the distribution of per job running time. Thus, T converges to a normal distribution with variance σ_T^2/n, where n represent the number of jobs, i.e. the number of samples. For a realistic workload, the variance σ_T^2 is fixed and finite. Thus, when batch interval increases, the increasing n on the denominator causes the decrease in variance. The same reasoning holds for a decrease in the variance for ε.

For brevity, we omit a formal derivation of the above results involving a lengthy expansion of T as the product of the number of jobs per batch and the per-job finishing time, followed by an application of the Central Limit Theorem to each component. The mathematical details are less important than the implication for MapReduce operators. Without changing the workload, i.e., σ_T^2 constant, or changing the underlying system, i.e., σ_ε^2 constant, we can decrease the per-batch running time variance simply by increasing the batch size.

Thus, for an increase in latency, MapReduce operators can obtain more predictable running times by decreasing the variance in the running time of the entire batch.

4.3 Performance Tradeoffs

The batching benefits above come at a cost of increased latency. Current methods to consider tradeoffs in multiple metrics are clumsy and fail to suggest a desirable tradeoff point. For example, we would compare 5-minutes batches with active fraction of mean 0.4 and variance 0.2 and latency of mean 400s and variance 200s to 15-minutes batches with active fraction of mean 0.2 and variance 0.1 and latency of mean 1000s and variance 100s. This comparison considers two vectors in four dimensions, with each performance vector being better in some dimension and not some others. MapReduce operators choosing between 5-minutes or 15-minutes batches would be compelled to use some heuristics to make a decision. Such heuristics become unusable when we consider the full CDF of performance in many dimensions.

Thus, we need a mechanism to jointly consider the effects of multiple performance dimensions such as latency and utilization. Further, we must account for any aversion to variance in either dimension. Fortunately, we do not need to invent such a mechanism. We borrow an established formulation from economic game theory - the Von Neumann-Morgenstern (VNM) utility function.

We use the VNM utility function for its two key properties: (i) it is additive and tractable, and (ii) it incorporates variance aversion in multiple dimensions. This formulation allows us to find tradeoff points systematically. Also, as will be evident below, selecting the particular VNM utility function involves an ad-hoc, heuristical component. We hope that our formulation and examples here serve as a catalyst for further discussion regarding multi-dimensional performance comparisons.

In the following section, we introduce a simplified formulation of VNM utility. We then provide a concrete example of how MapReduce operators can use this mechanism to select the right batch interval with a good tradeoff between latency and resource utilization.

4.3.1 VNM Utility in Brief

We demonstrate the VNM utility function using a simple example. For a given performance metric, define the best-case outcome a and worst-case outcome b. The utility at a certain outcome c is equivalent to a lottery that yields outcome a with probability p and outcome b with probability $1 - p$. In this example, a and b have VNM utility of 1 and 0 respectively. We next describe the key properties of a VNM utility function.

The "additive" property: A significant advantage of VNM utility is its additive property. This property implies that the utility of a collection of probabilistic events is its expected utility. In other words, if outcome c_i occurs with some probability q_i, then the utility of the collection of these outcomes is $\sum q_i U(c_i)$, where $U(c)$ is the utility function.

Given the additive property, the utility of a CDF is straight-forward. We multiply the utility at each point in the CDF with the probability associated with that point. We can divide the CDF of performance metric X into Q regular quantiles with quantile values represented by X_i, $i = 1, 2, ...Q$, with Q being arbitrarily large. The expected utility of the CDF is the expected value of $U(X_i)$, equal to $\frac{1}{Q}\sum U(X_i)$.

Variance aversion: VNM utility automatically incorporates variance aversion through the lottery formulation. If a system is neutral with regard to variance, its VNM utility would be a straight line from $(a, 1)$ to $(b, 0)$. If a system is variance averse or variance preferential, its VNM utility would be respectively a concave or convex line, respectively above or below the variance neutral line. For example, Figure 9 shows three possible VNM utility functions for latency. Consider the VNM utility at a latency of 300s. If the system is variance averse, it would equate 300 for sure with a lottery with expected latency value much lower than 300s. Thus, the variance averse line is above the variance neutral line. The identical analysis would show that the variance preferred line is below the variance neutral line. The stronger the aversion or preference for variance, the further the VNM utility function would deviate from the variance neutral line.

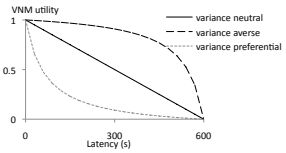


Figure 9: Typical utility functions.

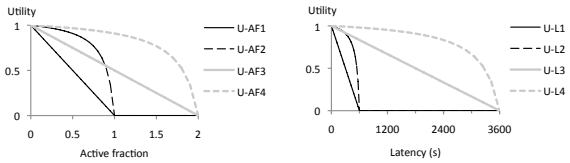


Figure 10: Utility functions used for performance comparisons.

Additive utility in many dimensions: We guarantee additivity across many dimensions if we formulate the utility in each using the same lottery. For example, we could formulate the utility of latency and resource utilization both in terms of lotteries in latency, or resource utilization, or a third metric.

Connection to MapReduce workload management: On the surface, the choice of VNM utility functions seems arbitrary, and its relevance to MapReduce workload management unclear. However, as the following section illustrates, the numerical value of the utility matters less than the rankings in the utility values. Such rankings allow MapReduce operators to identify good operating points in a multi-dimension performance tradeoff space. The method is more systematic than ad-hoc comparisons between performance vectors, and scales to a large number of dimensions.

4.3.2 Example - finding the right batch interval

Using the VNM approach involves three steps: defining a family of utility functions, computing expected utilities to rank tradeoff points, and interpreting the rankings.

Family of utility functions: While a single utility function may appear arbitrary by itself, a family of utility functions would allow MapReduce operators to explore different degrees of variance tolerance and ranges of acceptable values. We seek to rank expected utilities of the four batch sizes using a family of utility functions. Consistent ranking across different utility functions would build confidence that there is one winning tradeoff point.

Specifically, we use the utility functions in Figure 10. Utility functions for active fraction are labeled U-AF1 to U-AF4, while utility functions for latency are labeled U-L1 to U-L4. We consider four pairs of utility functions. (U-AF1, U-L1) and (U-AF2, U-L2) respectively represent variance neutral and variance averse systems that demand active fraction below 1.0 and latency below 600s. (U-AF3, U-L3) and (U-AF4, U-L4) are the counterparts for less demanding systems requiring active fraction less than 2.0 and and latency below 3600s.

We eliminate hourly and daily batch sizes as these batch sizes contribute zero utility per U-L1 to U-L4. The marginal improvement in active fraction going from 15-minutes to hourly and daily batches does not compensate for the latency penalty.

Table 1: Expected utility for different batch sizes. The AF columns show expected utility of active fraction CDFs, Lat columns show expected utility of latency CDFs, and Sum column adds the AF and Lat columns.

Facebook						
	5min			15 min		
Util. functions	AF	Lat	Sum	AF	Lat	Sum
U-AF1, U-L1	0.82	0.40	1.23	0.90	0.00	0.90
U-AF2, U-L2	0.98	0.88	1.85	0.98	0.00	0.98
U-AF3, U-L3	0.91	0.90	1.81	0.94	0.72	1.66
U-AF4, U-L4	0.99	0.99	1.98	0.98	0.96	1.94
Bimodal						
	5min			15 min		
Util. functions	AF	Lat	Sum	AF	Lat	Sum
U-AF1, U-L1	0.64	0.32	0.96	0.62	0.00	0.62
U-AF2, U-L2	0.86	0.76	1.62	0.91	0.00	0.91
U-AF3, U-L3	0.79	0.88	1.67	0.81	0.65	1.46
U-AF4, U-L4	0.95	0.99	1.93	0.97	0.95	1.92

Computing expected utility: We apply the utility functions in Figure 10 to the active fraction CDFs in Figure 8 and corresponding latency CDFs. Table 1 shows expected utility values for various batch interval sizes.

As per data in Figure 8, the utility of active fractions tend to increase with batch interval size. At large batch intervals, higher utility of active fraction is tempered by lower utility from increased latency. All utility functions suggest that 5 minutes is the preferred batch interval.

The VNM utility formulation also illustrates why batching is preferred to executing jobs as they arrive. If we do not batch, then the active fraction is always 1, and the latency is always 0. For these values, the sum utility are 1, 1, 1.5, and 1.9 for utility function pairs 1 to 4, regardless of the workload. Compared with Table 1, the 5 minute batch is again preferred for all utility functions.

Utility functions to design decisions: Our computations show that multiple utility functions rank the 5-minutes batch ahead of alternate batch sizes. The consistent ranking builds confidence that the ranking is not sensitive to different degrees of variance tolerance and ranges of acceptable value. It suggests to MapReduce operators that batching every 5-minutes is a good operating mode, provided that the active fraction and latency capture all performance dimensions of interest. If not, we can define VNM utility functions for other performance metrics, and re-rank the additive expected utility.

Several caveats apply when interpreting VNM utility

values. Our batch size selection process depends more on the relative utility values than the numerical values themselves. Thus, the VNM utility is not a definitive indication of monetary or intangible "value" to MapReduce operators; it merely helps decide between two performance tradeoff points.

Also, had the rankings been inconsistent, the preference between the tradeoff points would be sensitive to the choice of utility functions. In such cases, MapReduce operators can examine alternate design choices that lead to clearer tradeoffs or consider tradeoffs along additional performance dimensions.

4.4 Other Tradeoffs using VNM Utility

The previous section introduced the VNM utility formulation and used it to find a batch interval size that balances latency and resource utilization. We next use the VNM utility approach to examine two additional design choices - cluster size and machine type.

These examples serve a dual purpose. First, the measurement and comparison results help inform MapReduce operators about cluster configuration. It turns out that in both cases, the performance tradeoffs are clear enough that we can rank design choices even without the VNM utility formulation. Thus, the second purpose of these experiments is to verify the correctness of the VNM utility method when clear interpretations exist.

4.4.1 Cluster size

There are two ways to operate MapReduce on a large set of machines. We could run an entire workload on a single, large cluster or partition the workload and machines to run workload subsets on subsets of machines. Large clusters may be necessary to accommodate workload dataset size. However, data size permitting, smaller cluster may avoid cluster size scalability bottlenecks.

To investigate the tradeoff between large and small clusters, we run the Facebook and bimodal workloads on clusters of 10 m1.large machines on EC2, and compare the performance measurements with our preceding results on clusters of 100 m1.large machines. Recall that the SWARM scales the data size according to cluster size. Thus, the Facebook and bimodal workloads on a 10-node cluster would be computing over one-tenth the data on a 100-node cluster.

Figure 11 shows the active fraction CDF for the two workloads. Relative to Figure 8, the CDFs are near identical. This result empirically supports the belief that Hadoop easily scales to hundreds of nodes. In other words, a single large cluster of 100s of machines is equivalent to many small clusters of 10s of machines.

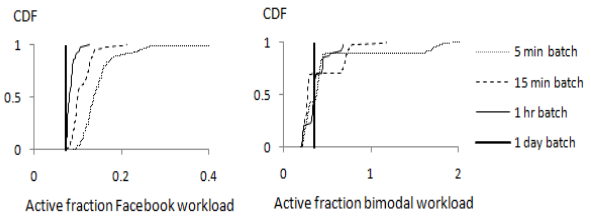


Figure 11: Active fractions for a cluster of 10 machines.

Table 2: Expected utility for a cluster of 10 machines. Condensed form of Table 1

	Facebook		*Bimodal*	
Util. functions	5min Sum	15 min Sum	5min Sum	15 min Sum
U-AF1, U-L1	1.28	0.89	0.90	0.60
U-AF2, U-L2	1.86	0.99	1.60	0.90
U-AF3, U-L3	1.83	1.67	1.64	1.45
U-AF4, U-L4	1.98	1.96	1.92	1.92

Table 2 shows the corresponding VNM analysis, again limited to the 5-minutes and 15-minutes batches. Compared with Table 2, the values are near identical. Thus, the comparison under the VNM formulation agrees with the comparison of active fraction CDFs. In particular, 5-minutes batches is preferred for both cluster sizes.

4.4.2 Machine types

Aside from cluster size, MapReduce operators can change the machine types in a cluster. In particular, for EC2, one can easily change a cluster of m1.large machine instances to a cluster of m1.small instances. For MapReduce users of EC2, it is important to understand which instance type is more cost efficient. In general, MapReduce operators make similar assessments on the cost-performance tradeoff of different cluster hardware to inform hardware purchasing choices for cluster upgrades or construct clusters of heterogenous machines.

To understand cost-performance tradeoffs among machine types, we run the Facebook and bimodal workloads on both a cluster of 50 m1.large machines and a cluster of 200 m1.small machines. A m1.large machine represents four times the CPU capacity of a m1.small machine, costing 4 times more. For EC2, cost and running time are proxies for each other. For simplify analysis, we assume that billing is at a second granularity instead of hourly granularity. Also, for fair comparison, we do not rescale the data size between the two clusters.

Figure 12 shows the active fraction CDF for both workloads across the two instance types. For the Facebook workload, m1.large cluster shows a narrower and lower CDF, with the 5-minute batch showing a more prominent decrease. In this case, the task slots are not saturated, and we have a fair comparison suggesting that

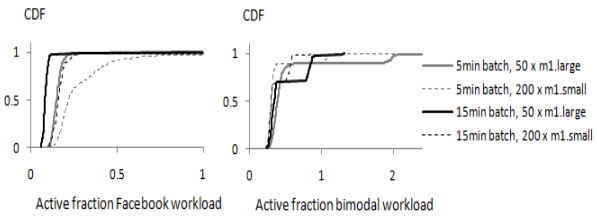


Figure 12: Active fractions for a cluster of 10 machines.

Table 3: Expected utility for cluster of 200 m1.small machines and cluster of 50 m1.large machines.

Facebook				
	5min		15min	
Util. functions	m1.small	m1.large	m1.small	m1.large
U-AF1, U-L1	1.07	1.27	0.83	0.91
U-AF2, U-L2	1.75	1.86	0.97	0.98
U-AF3, U-L3	1.73	1.80	1.61	1.68
U-AF4, U-L4	1.95	1.89	1.95	1.96
Bimodal				
	5min		15min	
Util. functions	m1.small	m1.large	m1.small	m1.large
U-AF1, U-L1	0.95	0.83	0.63	0.52
U-AF2, U-L2	1.62	1.57	0.94	0.89
U-AF3, U-L3	1.70	1.60	1.47	1.39
U-AF4, U-L4	1.96	1.87	1.93	1.91

the m1.large is more cost efficient.

For the bimodal workload, the reverse is observed. As m1.large instances are intended to have high I/O performance, this surprising result for the bimodal workload alludes to a cluster configuration issue. We constructed the bimodal workload to be more data intensive with smaller inter-job arrival times and larger data size. Even the "small" jobs in the bimodal workload have larger data sizes than most jobs in the Facebook workload. Larger jobs imply more map and reduce tasks under default Hadoop settings. Default Hadoop settings assign the same number of task slots to each machine, regardless of the machine size. Thus, in a cluster of fewer large machines, the number of task slots becomes a bottleneck. For the bimodal workload, having more small machines increases the availability of task slots to execute more map and reduce tasks in parallel.

This finding is significant. If possible, MapReduce operators should configure larger machines to have more task slots per machine. However, if MapReduce operators cannot control configuration, as when using EC2, they can check if the workload saturates available task slots. If so, it is cost-effective to move data-intensive workload to a cluster of many low I/O machines.

The corresponding VNM analysis is in Table 3. Again, the comparison under the VNM formulation agrees with the comparison in active fraction CDFs. The ranking is consistent across all four utility functions.

4.5 Multi-Tenant Workloads

Thus far, our experiments have focused on a single workload at a time. The natural next step is to look at multi-tenant workloads, i.e., concurrently running the Facebook and bimodal workloads on the same cluster. Multi-tenant workloads facilitate the statistical multiplexing of available computing resources. However, we must eliminate potential counter-productive interference. In addition, multi-tenancy highlights an inherent tension between MapReduce operators, who want high utilization and MapReduce users, who want low latency. These conflicting goals are identical to the latency-utilization tradeoff associated with batch execution, and we believe the VNM formulation can once again assist.

To understand multi-tenant performance tradeoffs, we ran the Facebook and bimodal workloads together on the same Hadoop cluster on EC2. This cluster has 50 m1.large machines, and is identical to the m1.large cluster of the machine instance type experiments. The multi-tenant performance baseline would be the CDF of the sum of Facebook and bimodal active fractions. There are three possible outcomes: (i) if there is interference, the multi-tenant CDF would be higher than the baseline CDF, and the variation would be larger, (ii) if there is statistical multiplexing, the multi-tenant CDF would be lower than the baseline CDF, and the difference preserved across different batch sizes, and (iii) if neither occurs, multi-tenancy is equivalent to a batch with more jobs so the multi-tenant CDF would be lower but the difference would significantly decrease at larger batch sizes.

Figure 13 shows the active fraction CDF for the multi-tenant workload, as well as the baseline CDF of the sum of Facebook and bimodal active fractions. Clearly, multi-tenancy leads to lower active fractions. To verify whether this decrease comes from statistical multiplexing of resources or more jobs in a batch, we compute the ratio between the multi-tenant CDF and the baseline CDF. For 5-minutes batches, the ratio is roughly 0.65. For 15-minutes batches, the ratio is marginally higher at 0.70. Thus, we observe a combination of statistical multiplexing and increased per-batch job count. Repeating the comparison at hourly and daily batch sizes would quantify the asymptotic ratio between multi-tenant and baseline running times as batch sizes get arbitrarily large.

We again use the VNM utility functions in Figure 10 to rank the latency-utilization tradeoffs. Table 4 shows the expected utility for the multi-tenant workload.

Comparing the sum of expected utilities with the m1.large columns in Table 3, we see that there is no consistent advantage to running workloads in isolations over running them in a multi-tenant fashion. The implication is that multi-tenancy consolidates, on a single cluster, workloads intended for two clusters. The unused cluster

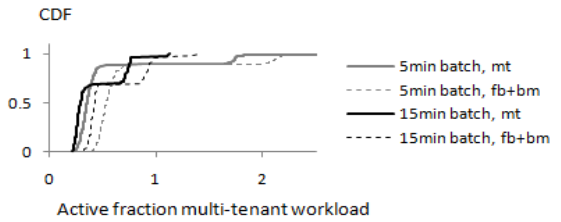


Figure 13: Active fractions for multi-tenant (mt) workload. Also showing the baseline CDF of the sum of Facebook and bimodal active fractions (fb+bm).

Table 4: Expected utility for the multi-tenant workload at 15-minutes and 5-minutes batches.

	5min			15 min		
Util. functions	AF	Lat	Sum	AF	Lat	Sum
U-AF1, U-L1	0.58	0.29	0.87	0.57	0.07	0.64
U-AF2, U-L2	0.85	0.74	1.58	1.34	0.33	1.66
U-AF3, U-L3	0.75	0.87	1.62	0.79	0.81	1.60
U-AF4, U-L4	0.93	0.99	1.92	0.97	0.98	1.95

is now available for other uses. Our measurements and the VNM utility analysis show that multi-tenancy can be achieved with manageable cost.

There are several additional design considerations for multi-tenant workload management, including the choice between priority-based and deadline-driven schedulers. MapReduce operators can further explore tradeoffs between consolidating many different workloads on a shared cluster and consequent performance degradation. These design considerations require analysis of multiple performance metrics due to a variety of workload parameters and heuristic-based approaches for decision making become intractable. The VNM utility formulation offers a succint way to jointly consider such tradeoffs among many dimensions.

5 Related Work

Understanding performance tradeoffs in MapReduce-style computations requires a systematic approach to isolate the sources of performance variation. To this end SWARM differs from, but in some cases, complements preceding efforts in four key areas:

MapReduce benchmarks and simulators: The earliest MapReduce performance evaluations use sort and grep type jobs as microbenchmarks. The gridmix pseudo-benchmark represents an improvement on sort [1] by its inclusion of five different jobs, each with different data ratios, that run on data sizes ranging from 500GB to a few TB. While gridmix contains more job types, it is still does not capture production workload variability. In contrast, we bootstrap SWARM with real traces and employ statistical techniques to synthesize workloads with representative characteristics along many workload dimensions.

The Mumak Hadoop simulator seeks to replicate detailed behavior of a Hadoop cluster [3]. Mumak expedites evaluation and debugging of new software mechanisms but requires the full data layout information. As such our techiques for analyzing production traces and synthesizing representative workloads will facilitate more realistic evaluations in Mumak with less overhead.

Workload generation and replay tools: Current workload generation tools suffer two key limitations - they rely on pre-defined statistical patterns, and/or the joint generation and replay mechanisms do not scale.

Several application-specific tools only generate workloads that conform to a parametric pattern, e.g., SURGE for URLs [9], SLAMD for LDAP servers [5], StreamGen for data streams [18], and Harpoon for Internet traffic [19]. SWARM, in contrast, samples empirical CDFs and PDFs of various job-characteristics in production traces to construct workloads and thus also handles non-parametric distributions.

Many web service workload generators have transformed into benchmarks [6, 7, 4] but suffer from two limitations. They (i) use stationary matrices of transition probabilities for request generation and (ii) scale poorly due to per-user/request state maintenance overhead. SWARM separates workload synthesis from workload replay/execution. Consequently, we can use resource intensive techniques for generating our job stream. SWARM also requires less per-job state and thus seamlessly scales to very high job event rates.

Isolating sources of performance variability: The database community has studied performance variability due by workload as well as inherent to the system itself.

Prior work on adaptive query processing modifies query execution based on the systems' utilization and performance characteristics including I/O latencies and data transfer rates [14]. These techniques have been adapted for Internet applications by additionally considering service level objectives and tradeoffs between incomplete results and query completion time [17].

There is sizeable literature on understanding variance in the underlying data and consequent performance predictability of parallel database workload. The petabyte scale of data magnifies variance and imposes high penalties for inaccurate data cardinality estimates [16].

The experiments in this paper distinguish between three MapReduce performance variability sources.

Comparing and ranking performance metrics: Direct predecessors to our work, [12] and [10] examine MapReduce configuration decisions and compression tradeoffs respectively. [15] uses multi-dimensional input characteristics for accurate performance prediction, demonstrating the advantage of combining multiple performance dimensions. Using VNM utility functions, we

build on prior multi-dimensional modeling work, formalize tradeoffs between variability and performance, and systematically derive a reasonable operating point.

6 Conclusions and Future Work

Our experimental results address several MapReduce operational considerations. First, there is an inherent tradeoff between latency and utilization due to conflicting priorities between cloud operators and application developers. Next, in the absence of representative workloads, operators must understand statistical properties of their workloads to optimize appropriately.

In this paper, we show snapshots of two production Hadoop workloads. These traces inspired us to develop the SWARM workload analysis and replay tool. We used SWARM to generate two synthetic workloads, one mimicing Facebook traces and the other explicitly to artificially challenge our design decisions. We replayed both workloads on EC2 to investigate tradeoffs in batching, cluster size, machine types, and multi-tenant workload management. We introduced the VNM utility formulation to additively capture tradeoffs in multiple performance dimensions.

The SWARM tool demonstrates that even for applications as challenging as MapReduce, it is possible to synthesize realistic workloads by computing statistical distribution from production traces, and sampling these distributions. This method generalizes to arbitrary traces such that the more workload characteristics the trace captures, the more realistic the generated workload can be. More importantly, SWARM can be generalized to other (non-MapReduce) applications by merely changing the traces used for bootstrapping. Thus, we believe the SWARM approach is powerful for many cloud computing applications.

Additionally, our VNM utility formulation allow us to consider multi-dimensional performance preferences in general, in addition to the utilization-latency tradeoff at the center of several design choices. We demonstrate that a family of utility functions can capture different degrees of variance aversion, and multiple definitions of "good" performance. Given the distributed nature of cloud applications, any design and policy decisions should consider tradeoffs along multiple dimensions. We believe the VNM utility formulation offers a tractable method to explore the complex performance space.

Our methodology generalizes to other MapReduce traces as well as non-MapReduce applications. Thus, the natural next step is to apply our methodology to contexts involving non-MapReduce applications. This could be single application environments such as distributed storage, or multiple application environments such as clusters concurrently running interactive web apps, MapReduce data mining, and development and testing workloads.

References

[1] Gridmix. HADOOP-HOME/src/benchmarks/gridmix in all recent Hadoop distributions.

[2] Hadoop. http://hadoop.apache.org.

[3] Mumak. http://issues.apache.org/jira/browse/MAPREDUCE-728, last retrieved Nov. 2009.

[4] RUBiS: Rice University Bidding System. http://rubis.objectweb.org.

[5] Slamd. http://www.slamd.com.

[6] SPECweb. http://www.spec.org/web2005.

[7] TPC-W. www.tpc.org/tpcw/default.asp.

[8] APACHE SOFTWARE FOUNDATION. Hadoop On Demand. http://hadoop.apache.org/common/docs/r0.18.2/hod.html.

[9] BARFORD, P., AND CROVELLA, M. E. Generating Representative Web Workloads for Network and Server Performance Evaluation. In *Proceedings of Performance '98/SIGMETRICS '98* (July 1998), pp. 151–160.

[10] CHEN, Y., GANAPATHI, A., AND KATZ, R. To compress or not to compress - compute vs. io tradeoffs for mapreduce energy efficiency. Tech. rep., UC Berkeley, 2010.

[11] CHEN, Y., GANAPATHI, A. S., FOX, A., KATZ, R. H., AND PATTERSON, D. A. Statistical workloads for energy efficient mapreduce. Tech. Rep. UCB/EECS-2010-6, EECS Department, University of California, Berkeley, Jan 2010.

[12] CHEN, Y., KEYS, L., AND KATZ, R. H. Towards energy efficient mapreduce. Tech. Rep. UCB/EECS-2009-109, EECS Department, University of California, Berkeley, Aug 2009.

[13] DEAN, J., AND GHEMAWAT, S. MapReduce: Simplified Data Processing on Large Clusters. *Communications of the ACM 51*, 1 (January 2008), 107–113.

[14] DESHPANDE, A., HELLERSTEIN, J. M., AND RAMAN, V. Adaptive query processing: why, how, when, what next. In *SIGMOD Conference* (2006), pp. 806–807.

[15] GANAPATHI, A., CHEN, Y., FOX, A., KATZ, R., AND PATTERSON, D. Statistics-driven workload modeling for the cloud. In *In press. SMDB '10: International Workshop on Self Managing Database Systems* (2010).

[16] GANAPATHI, A., KUNO, H., DAVAL, U., WIENER, J., FOX, A., JORDAN, M., AND PATTERSON, D. Predicting Multiple Performance Metrics for Queries: Better Decisions Enabled by Machine Learning. In *Proc International Conference on Data Engineering* (2009).

[17] IVES, Z. G., LEVY, A. Y., WELD, D. S., FLORESCU, D., AND FRIEDMAN, M. Adaptive query processing for internet applications. *IEEE Data Engineering Bulletin 23* (2000), 200–0.

[18] MANSOUR, M., WOLF, M., AND SCHWAN, K. Streamgen: A workload generation tool for distributed information flow applications. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 55–62.

[19] SOMMERS, J., KIM, H., AND BARFORD, P. Harpoon: a flow-level traffic generator for router and network tests. *SIGMETRICS Perform. Eval. Rev. 32*, 1 (2004), 392–392.