# Kahuna: Problem Diagnosis for MapReduce-Based Cloud Computing Environments

Jiaqi Tan, Xinghao Pan
DSO National Laboratories, Singapore
Singapore 118230
{tjiaqi,pxinghao}@dso.org.sg

Eugene Marinelli, Soila Kavulya, Rajeev Gandhi
and Priya Narasimhan
Electrical & Computer Engineering Dept.
Carnegie Mellon University, Pittsburgh, PA 15213
emarinel@andrew.cmu.edu,{spertet,rgandhi}@ece.cmu.edu, priya@cs.cmu.edu

***Abstract*—We present Kahuna, an approach that aims to diagnose performance problems in MapReduce systems. Central to Kahuna's approach is our insight on peer similarly, i.e., that nodes behave alike in the absence of performance problems, and that a node that behaves differently is the likely culprit of a performance problem. Kahuna incorporates techniques to statistically compare black-box (OS-level performance metrics) and white-box (Hadoop-log statistics) data across the different nodes of a MapReduce cluster, in order to identify the faulty node(s). We motivate our peer-similarity observations through concrete evidence from the 4000-processor Yahoo! M45 Hadoop cluster. In addition, we demonstrate Kahuna's effectiveness through experimental evaluation of its algorithms for a number of reported performance problems, on four different workloads (including `Nutch` and `Pig`) in a 100-node Hadoop cluster hosted on Amazon's EC2 datacenter.**

## I. Introduction

Cloud computing is becoming increasingly common, and has been facilitated by frameworks such as Google's MapReduce [1], which parallelizes and distributes jobs across large clusters. Hadoop [2], the open-source implementation of MapReduce, has been widely used at large companies such as Yahoo! and Facebook [3] for large-scale data-intensive tasks such as click-log mining and data analysis. Performance problems–faults that cause jobs to take longer to complete, but do not necessarily result in outright crashes–pose a significant concern because slow jobs limit the amount of data that can be processed. Commercial datacenters like Amazon's Elastic Compute Cloud (EC2) charge $0.10-0.80/hour/node, and slow jobs impose financial costs on users. Determining the root cause of performance problems and mitigating their impact can enable users to be more cost-effective.

Diagnosing performance problems in MapReduce environments presents a different set of challenges than multi-tier web applications. Multi-tier web applications have intuitive time-based service-level objectives (SLOs) as they are required to have low latency. Current state-of-the-art problem-diagnosis techniques in distributed systems rely on knowing which requests have violated their SLOs and then identify the root-causes [4] [5] [6]. However, MapReduce jobs are typically long-running (relative to web-request processing), with Google jobs averaging 395 seconds on 394-node clusters [7], or equivalently, 43 node-hours (i.e., with a 43-node cluster, the average job will run for an hour). These job times are dependent on the input size and the specific MapReduce application. Thus, it is not easy to identify the "normal" running time of a given MapReduce job, making it difficult for us to use time-based SLOs for identifying performance problems.

In the Kahuna technique, we determine if a performance problem exists and identify the culprit nodes, in a MapReduce system, based on the key insight of *peer-similarity* among nodes in a MapReduce system: (1) the nodes (that we loosely regard as "peers") in a MapReduce cluster tend to behave symmetrically in the absence of performance problems, and (2) a node that behaves differently from its peer nodes, is likely to be the culprit of a performance problem[1]. In this paper, (1) we evaluate the extent to which the peer-similarity insight is true on Hadoop, the most widely-used open-source MapReduce system, based on empirical evidence from data from real-world research jobs on the 4000-processor, 1.5 PB M45 cluster, a Yahoo! production cluster made available to Carnegie Mellon researchers, and (2) we investigate the extent to which this insight can be used to diagnose performance problems. We experimentally evaluate of two of our earlier problem-diagnosis algorithms based on the peer-similarity insight. These two algorithms diagnose problems in Hadoop clusters by comparing black-box, OS-level performance metrics [8], and white-box metrics derived from Hadoop's logs [9], respectively, and are examples of algorithms that can be built around the peer-similarity insight. We refer to them as Kahuna-BB and Kahuna-WB respectively. We perform extensive evaluation of these algorithms using multiple workloads and realistically-injected faults. Concretely, our contributions in this paper are: (1) empirical evidence to substantiate the validity of our peer-similarity insight, and (2) extensive experimental evaluation of whether the peer-similarity insight can be built on to diagnose performance problems.

## II. Background: MapReduce & Hadoop

Hadoop [2] is an open-source implementation of Google's MapReduce [7] framework that enables distributed, data-intensive, parallel applications by decomposing a massive job into smaller (map and reduce) tasks and a massive data-set

[1]We do not claim the converse: we do not claim that a performance problem will necessarily result in an asymmetrically behaving node. In fact, we have observed correlated performance degradations for certain problems, whereby all of the nodes behave identically, albeit incorrectly.

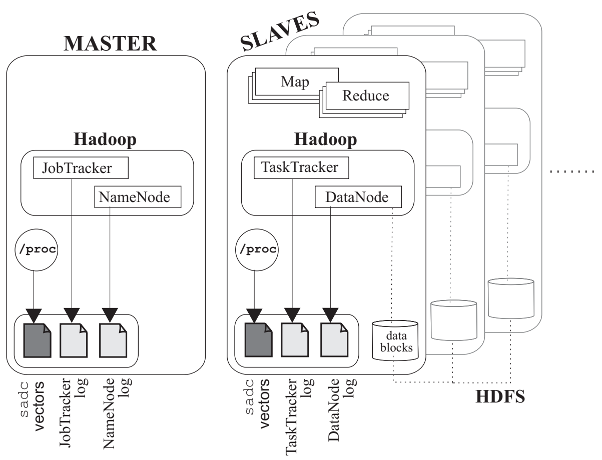

Fig. 1. Architecture of Hadoop, showing the logs of interest to Kahuna.

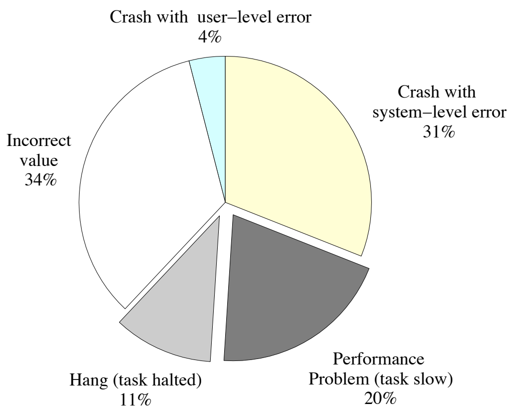

Fig. 2. Manifestations of Hadoop bugs reported in Jan 2007–Feb 2008.

into smaller partitions, such that each task processes a different partition in parallel. Hadoop uses the Hadoop Distributed File System (HDFS) implementation of the Google Filesystem [10], to share data amongst the tasks in the system. HDFS splits and stores files as fixed-size blocks. Hadoop uses a master-slave architecture, as shown in Figure 1, with a unique master node and multiple slave nodes. The master node typically runs the JobTracker daemon which schedules tasks belonging to a running job and the NameNode daemon which manages the HDFS regulates access to files by clients (i.e., the executing tasks). Each slave node runs the TaskTracker daemon which launches and tracks tasks on its local node, as directed by the remote JobTracker , and the DataNode which serves data blocks (on its local disk) to HDFS clients. Figure 1 shows the inputs to Kahuna-BB, `/proc`-based OS performance data (`sadc`-vectors), and to Kahuna-WB, Hadoop's TaskTracker and DataNode logs, respectively.

## III. Problem Statement

Our primary research question focuses on whether a peer-similarity based approach can localize performance problems accurately, even in large clusters. Ancillary questions concern the analysis of black-box *vs.* white-box data, and their respective fault-coverage with respect to their ability to capture the symptoms of various performance problems.

**Motivating evidence.** To investigate the impact of performance problems on Hadoop, we studied the manifestations of reported bugs from the Hadoop bug database [11] over a 14-month period from Jan 2007 to Feb 2008. We found that 31% of bugs manifested as degraded performance (hangs, performance problems), and a further 35% of bugs manifested as crashes (see Figure 2), indicating that performance problems are an important class of problems to address for diagnosis. Furthermore, we believe that the impact of performance problems is underrepresented in the bug database, as those which do not crash nor halt Hadoop are likely to go unreported.

**Goals.** Our diagnosis should run transparently to, and not require any modifications of, both the hosted applications and Hadoop itself. Our diagnosis should be usable in production environments, where administrators might not have the luxury of instrumenting applications for code-level debugging, but could likely leverage readily available black-box data or existing native Hadoop logs. Our diagnosis should produce *low false-positive rates*, in the face of a variety of workloads for the system under diagnosis, even if these workloads fluctuate, as in the case of real-world workloads such as `Nutch` and `Pig`. Our data-collection should impose minimal instrumentation overheads on the system under diagnosis.

**Fault model.** Performance problems in Hadoop clusters can result in jobs taking longer than usual to complete. Such abnormally long runtimes can be due to environmental factors on the node (e.g., an external, non-Hadoop workload), due to causes not specific to the node (e.g., non-deterministically triggered bugs within Hadoop), or due to data-dependent issues. The outcome of our diagnosis algorithm is a set of slave nodes in the cluster that our algorithm has identified as those that caused the job to take longer than usual to complete. We do not attempt to predict the normal completion-time of a given job. Instead, we identify jobs that undergo performance problems that cause the jobs to take longer to complete than if the problem were not present, and identify the nodes that caused the slowdown.

**Assumptions.** We assume that the target MapReduce system (i.e., the Hadoop infrastructure and Hadoop jobs) is the dominant source of activity on every node. We assume that a majority of the Hadoop slave nodes are problem-free and that all nodes are homogeneous in hardware. As a part of future work (see Section X), we propose to study the peer-similarity principle in the case of heterogeneous cluster configurations; this is outside the scope of the current paper.

**Non-goals.** We currently aim for coarse-grained problem diagnosis that localizes problems to the culprit slave node(s). This differs from finer-grained root-cause analysis that might aim instead to identify the underlying fault or bug, or offending line of code. While the presented algorithms can be run online, we focus on the offline evaluation of collected data. We also currently do not target master node problems, nor attempt to predict normal runtimes of a given job.

## IV. Empirical Evidence of Peer-Similarity

Over the past year, we have conducted several experiments with, and analyzed data from, different large-scale MapReduce systems, including our 100-processor Amazon EC2-based Hadoop cluster [12] as well as the 4000-processor Yahoo! M45 production cluster [13]. We experimented with multiple workloads, such as the simple `RandWriter` and `Sort` and the more realistic `Nutch` and `Pig`. We have also analyzed traces of Hadoop jobs run by other researchers on M45.

We have observed that, in the absence of performance problems, the slave nodes in a MapReduce cluster tend to exhibit similar behavior, as measured in any number of ways, e.g., CPU usage, network traffic, Map-task completion-times, etc. When we inject a performance problem (or when we observe a problem in the field, in the case of the M45 traces), we observed further that the slave node on which the problem originated (the *culprit* node) deviated from the other slave nodes in its behavior. These observations held across multiple traces, multiple experiments, multiple users, and testbeds. These two empirically-observed insights–*peer similarity under fault-free conditions* and *peer-divergence indicates a performance problem*–form the key motivators of our Kahuna approach, as described in Section V.

**Justification.** This observed behavior is intuitive and reasonable to expect in a MapReduce cluster. Slave nodes typically process copies of the same task that operate on different inputs; with each task processing large numbers of records, the aggregate behavior of slave nodes that we observe would be similar. In addition, we argue that, for a job to complete as quickly as possible, nodes (with the same processing capabilities) should spend comparable amounts of time handling comparable amounts of work, as the system completes its job only when the slowest node completes. Hence, *peer-similarity* amongst (homongeous) nodes is not merely an observed, but also a necessary, characteristic, for optimal performance.

**Supporting evidence.** We provide empirical evidence from traces of real Hadoop jobs in the Yahoo! M45 production cluster, to support our key insights. We obtained the Hadoop logs for jobs that were run by researchers (whose data-intensive jobs range from machine translation to large-scale data mining) over the five-month period, April–September 2008. The logs spanned 12854 experiments run by 26 users. Each experiment consisted of multiple Hadoop jobs. Table I provides a summary of the log statistics. Apart from the visual inspection of these log traces, we sought to establish, in a statistical way, that (i) slave nodes performing the same jobs behave similarly, and (ii) the similarity between nodes is affected when a job suffers from performance problems.

We measured the similarity among nodes by computing the absolute value of the Pearson's pair-wise correlation coefficient (between 0 and 1, with 1 expressing maximum similarity) of the average number of `Maps`/second being executed across slave nodes during a job across every pair of nodes in each job. We considered the number of `Maps`/second, rather than the number of `Reduces`/second, because a third of the jobs (for this specific user and workload) did not involve any `Reduce` tasks. Figure 3 shows a graphical view of the number of `Maps`/second being executed on each node. We found that, for successfully completed jobs, 89% of the nodes in had correlation coefficients $\geq 0.6$ with all of other nodes involved in the job. On the other hand, for jobs aborted by the JobTracker, only 43% of the nodes with correlations $\geq 0.6$ with other nodes involved in the job. This evidence supports our insights of peer-similarity amongst slave nodes in fault-free conditions, while nodes behave much less similarly in jobs that failed to complete. This is illustrated visually in Figure 3.

TABLE I
Summary of M45 logs.

| | |
|---|---|
| **Number of experiments** | 12854 |
| **Number of users** | 26 |
| **Average jobs per experiment** | 3.84 $\pm$ 20 |
| **Average nodes per experiment** | 21.42 $\pm$ 24 |
| **Average experiment duration (mins)** | 19.74 $\pm$ 82 |
| **Longest running experiment (hours)** | 23.32 |

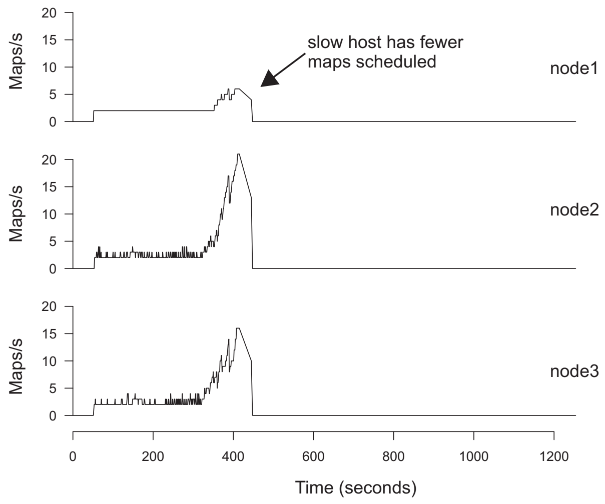

Fig. 3. Peer-comparison of `Map` tasks scheduled across M45 hosts helps to highlight a slow host in a user's Hadoop job.

## V. Diagnosis Approach

Based on our key insights in Section IV, we assert that a node whose behavior differs from the majority of nodes in the cluster is anomalous and can be a potential source of a performance problem. To enable us to quantify how similar/dissimilar nodes are to each other, we need to define the notion of "behavior" more concretely.Hence, Kahuna-BB statistically compares black-box data across nodes, and Kahuna-WB statistically compares white-box data across nodes.

**Black-box instrumentation.** We gather and analyze black-box (i.e., OS-level) performance metrics, without requiring any modifications to Hadoop, its applications or the OS. We use the `sadc` program in the `sysstat` package [14] to collect 14 OS metrics, as listed in Table II) from */proc*, at a sampling interval

| | |
|---|---|
| user | % CPU time in user-space |
| system | % CPU time in kernel-space |
| iowait | % CPU time waiting for I/O job |
| ctxt | Context switches per second |
| runq-sz | Number of processes waiting to run |
| plist-sz | Total number of processes and threads |
| ldavg-1 | system load average for the last minute |
| eth-rxbyt | Network bytes received per second |
| eth-txbyt | Network bytes transmitted per second |
| pgpgin | KBytes paged in from disk per second |
| pgpgout | KBytes paged out to disk per second |
| fault | Page faults (major+minor) per second |
| bread | Total bytes read from disk per second |
| bwrtn | Total bytes written to disk per second |

TABLE II
GATHERED BLACK-BOX METRICS (`sadc`-VECTOR).

of one second. We denote each vector containing samples of these 14 metrics, all collected at the same instant of time, `sadc`-vector. We use these `sadc`-vectors as our metric for diagnosis in Kahuna-BB.

**White-box instrumentation.** We collect the system logs generated by Hadoop's native logging code from the TaskTracker and DataNode daemons on slave nodes, and use the SALSA log-analysis technique [9] to extract state-machine views of the execution of each daemon. Kahuna-WB treats each entry in the log as an event, and uses particular events to identify states in the control-flow of each daemon, e.g., `Map` and `Reduce` states in the TaskTracker, and `ReadBlock` and `WriteBlock` states in the DataNode. Kahuna-WB generates sequences of state executions, data-flows between these states, and durations of each state, for each node in the cluster. Kahuna-WB compares these durations across slave nodes to perform its diagnosis.

## VI. DIAGNOSIS ALGORITHMS

For each of the metrics of diagnosis (`sadc`-vector for Kahuna-BB and state durations for Kahuna-WB), over a given period of time, we compare that metric's values at each node with the metric's corresponding values at all of the other nodes. If the values of metric *X* at a given node, *N*, differ significantly from sufficiently many nodes in the cluster, metric *X* is said to indict node *N* as the culprit, and our algorithms flag the presence of a performance problem. Kahuna-BB and Kahuna-WB are similar in their use of a peer-comparison strategy, but differ in that (i) they operate on different sets of metrics, (ii) they impose different hypotheses on the data, and (iii) they can operate independently of each other.

### A. Kahuna-WB: White-Box Diagnosis

We assume that state durations of interest have been extracted from Hadoop's logs using [9]. Kahuna-WB performs peer-comparison on the durations of the states extracted through log analysis, i.e. the `Map`, `Reduce`, ReduceCopy and ReduceMergeCopy states from TaskTracker logs, and the `ReadBlock` and `WriteBlock` states from DataNode logs. For each state, for a given period of time, we generate a histogram of the durations of that state for each node, in order to create an aggregate view of the state's durations for that node. Then, for each node, we compare its histogram for that state's duration with the corresponding histograms of every other node in the cluster. We perform this comparison by computing a statistical measure (square-root of the Jensen-Shannon divergence [15]) of the distance between histograms. If a given node's histogram differs significantly (by more than a threshold value) from those of a majority of the other nodes, then, we diagnose the cluster as undergoing a performance problem with that node being the culprit.

### B. Kahuna-BB: Black-Box Diagnosis

Unlike Kahuna-WB, which simply performs a statistical comparison across nodes, Kahuna-BB requires *a priori* training to summarize system metrics into a small number of behavior profiles. This enables it to extract meaning out of black-box data typically devoid of semantic information. Kahuna-BB is based on two related hypotheses about the behavior of MapReduce slave nodes from the perspective of black-box, OS-level performance counters: (i) that MapReduce slave nodes exhibit a small number of distinct behaviors, and that (ii) in a short interval of time (say, 1s), the system's performance tends to be dominated by one of these behaviors. We then exploit our peer-similarity insight in Kahuna-BB to incriminate any node whose dominant distinct behavior differs from those of a majority of the other nodes in the cluster.

**Learning.** The *K* distinct profiles, each of which captures one of the distinct Hadoop behaviors hypothesized above, are first learned *a priori* from fault-free runs of various workloads (described in Section VII-A). This phase involves collecting `sadc`-vectors from these runs, and then using the Expectation-Maximization (EM) algorithm [16] to cluster these `sadc`-vectors into behavior profiles. These profiles have no semantic meaning, and are used to condense Hadoop's different behaviors into a small number of profiles, as manifested across all 14 elements of the `sadc`-vectors.

**Diagnosis.** First, we classify each node's `sadc`-vector into one of the learned *K* distinct profiles. We then compute the histograms of these profiles for each node, decayed exponentially with time to weight past behavior less. We then compute the pairwise distance between the histograms of every pair of nodes in the cluster in each time period, using a statistical measure (the square-root of the Jensen-Shannon divergence [15]). If a specific node's histogram differs significantly (by more than a threshold value) from those of a majority of the other nodes, then we declare the system to be undergoing a performance problem and indict that node to be the culprit.

## VII. EXPERIMENTAL EVALUATION

Each experiment consists of a {problem, workload} pair, with a Hadoop cluster running a particular workload, with a specific single fault injected at one of the slave nodes during the experiment. These injected performance problems increase the workload's running time, as compared to the control fault-free case, as shown in Table IV. We evaluate Kahuna-BB's and Kahuna-WB's ability to detect and localize the injected

| Fault / Workload | `RandWriter` | `Sort` | `Nutch` | `Pig` |
|---|---|---|---|---|
| Control (none) | 0% | 0% | 0% | 0% |
| *CPUHog* | 7.2% | 28.2% | 25.1% | 11.6% |
| *DiskHog* | 5.7% | 26.6% | 10.6% | 16.3% |
| DiskFull | 1.4% | 4% | 0% | 0% |
| PacketLoss1 | 0% | 0% | 0% | 0% |
| PacketLoss5 | 0% | 23.2% | 30.1% | 0% |
| PacketLoss50 | 0% | 89.8% | 116.5% | 561.1% |
| HANG-1036 | 1% | 12.8% | 99.4% | 37.0% |
| HANG-1152 | 0% | 24.4% | 44.0% | 6.6% |
| HANG-2080 | 0% | 23.7% | 6.1% | 0.5% |

TABLE IV
IMPACT OF INJECTED PROBLEMS ON PERFORMANCE OF EACH WORKLOAD. WE REPORT THE PERCENTAGE SLOWDOWN OF THE COMPLETION TIME OF EACH JOB RELATIVE TO THE "CONTROL" CASE.

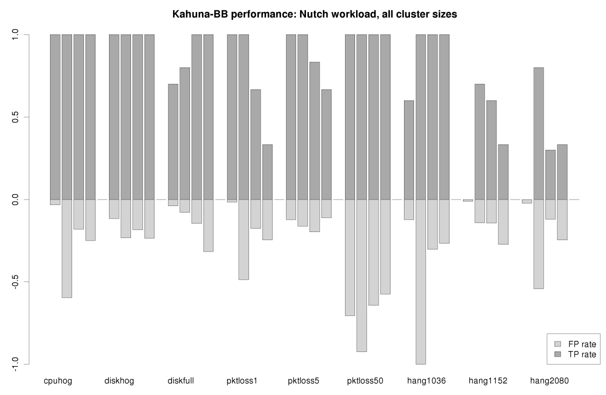

Fig. 4. Results for all problems on `Nutch` for all cluster sizes.

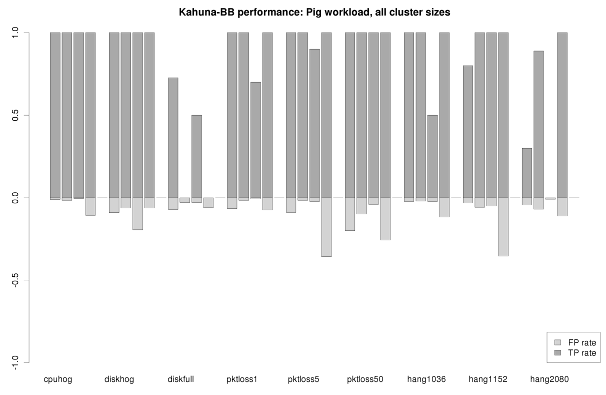

Fig. 5. Results for all problems on `Pig` for all cluster sizes.

performance problems for each Hadoop workload. We selected these injected problems to cover both (i) resource-contention (CPU, disk, network) problems, as well as (ii) Hadoop bugs that result in performance slowdowns or hangs.

## A. *Experimental Setup*

We collected and analyzed traces from Hadoop clusters running on virtual machines on the Amazon EC2 virtual datacenter. Each node consisted of an "extra-large" node instance on EC2, with 4 dual-core Intel Xeon-class CPUs, 15GB of memory, and 1.6TB of Elastic Block Store (EBS) storage [12]. We collected traces from 10-, 25-, 50- and 100-node Hadoop clusters. We collected both black-box, OS-level performance counters throughout the experiment, and white-box Hadoop logs at the end of each experiment for offline processing. Black-box data collection incurred an overhead of less than 1% CPU utilization; white-box data-collection did not require additional effort as Hadoop generates logs by default.

**Workloads.** To evaluate the effectiveness of Kahuna on a variety of workloads, we collected traces of Hadoop under four different workloads: two simple benchmarks that are packaged with Hadoop (`Sort` , `RandWriter`), and two Hadoop applications (`Nutch`, `Pig`) commonly used in real-world installations. These are described below:

- `RandWriter`: Write-intensive workload that generated 24 GB of random data on each task
- `Sort` : Read/Write workload that sorts 5 GB of randomly-ordered records generated by `RandWriter`. `Sort` and `RandWriter` are common Hadoop benchmarks
- `Nutch`: Distributed web-crawler for Hadoop, in use at several companies [3], representative of a real-world, fluctuating workload typical of many Hadoop workloads
- `Pig`: High-level language for expressing data analysis programs [17], that is compiled into Hadoop programs, typical of a sophisticated, multi-job Hadoop workload

**Fault Injection.** Table III describes the faults that we injected into one of the slave nodes in each experiment, and our fault-injection methodology. These faults trigger performance problems reported on the Hadoop users' mailing list or on the Hadoop bug database [11]. The faults studied were intentionally selected to induce performance problems, specifically those that caused Hadoop jobs to take longer to complete than expected, as compared with our *control* (i.e. fault-free) experiments. Table IV captures the impact of these injected faults on the job-completion times of the four workloads.

## VIII. RESULTS

For each experimental trace, we analyzed the trace-data using both Kahuna-WB and Kahuna-BB, each of which returned a list of nodes it indicted as culprits. We evaluated this outcome as follows: an indictment of the fault-injected node was a true-positive, an indictment of any other node was a false-positive (FP), and a failure to indict the fault-injected node was a false-negative (FN). The true-positive (TP) and false-positive rates ($\in [0.0, 1.0]$, with $TP = 1.0, FP = 0.0$ representing a perfect diagnosis) are presented here for 10 fault-induced experimental runs for each {problem, workload} pair. These results are computed for a specific threshold (for Kahuna-BB and Kahuna-WB) for each {workload, cluster-size} pair.

## A. *Kahuna-BB: Black-box diagnosis*

Figures 4, 5, 6, and 7 show the TP and FP rates for each {fault,workload} pair. The bar graphs are grouped by injected fault, and each set of 4 bars shows results for the 10-, 25-,

| [Source] Reported Failure | [Problem Name] Problem-Injection Methodology |
|---|---|
| [Hadoop users' mailing list, Sep 13 2007] CPU bottleneck resulted from running master and slave daemons on same machine | [CPUHog] Emulate a CPU-intensive task that consumes 70% CPU utilization |
| [Hadoop users' mailing list, Sep 26 2007] Excessive messages logged to file during startup | [DiskHog] Sequential disk workload wrote 20GB of data to filesystem |
| [HADOOP-2956] Degraded network connectivity between DataNodes results in long block transfer times | [PacketLoss1/5/50] 1%,5%,50% packet losses by dropping all incoming/outcoming packets with probabilities of 0.01,0.05,0.5 |
| [HADOOP-1036] Hang at TaskTracker due to an unhandled exception from a task terminating unexpectedly. The offending TaskTracker sends heartbeats although the task has terminated. | [HANG-1036] Revert to older version and trigger bug by throwing NullPointerException |
| [HADOOP-1152] Reduces at TaskTrackers hang due to a race condition when a file is deleted between a rename and an attempt to call getLength() on it. | [HANG-1152] Simulated the race by flagging a renamed file as being flushed to disk and throwing exceptions in the filesystem code |
| [HADOOP-2080] Reduces at TaskTrackers hang due to a miscalculated checksum. | [HANG-2080] Deliberately miscomputed checksum to trigger a hang at reducer |

TABLE III
INJECTED PROBLEMS, AND THE REPORTED FAILURES THAT THEY SIMULATE. HADOOP-XXXX REPRESENTS A HADOOP BUG-DATABASE ENTRY.

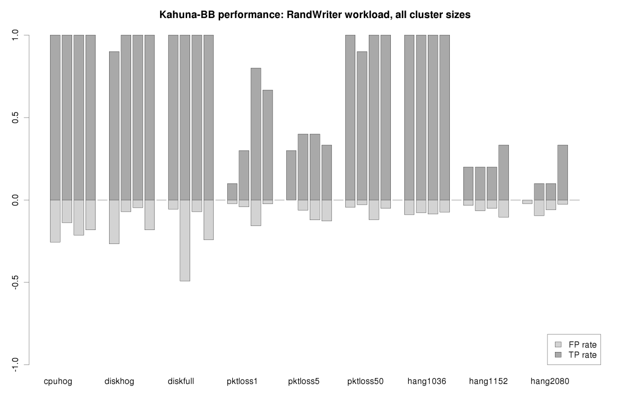

Fig. 6. Results for all problems on `RandWriter` for all cluster sizes.

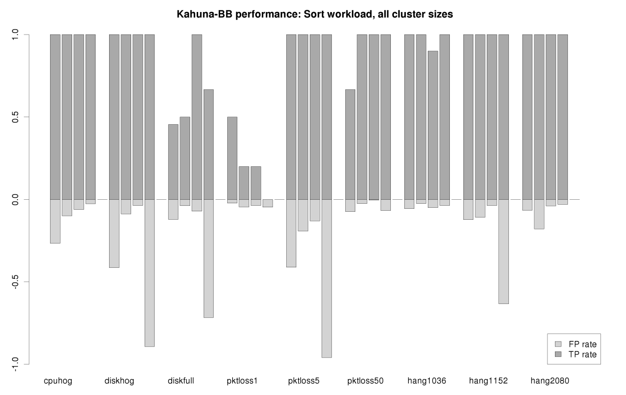

Fig. 7. Results for all problems on `Sort` for all cluster sizes.

50-, and 100-node clusters respectively. In general, Kahuna-BB was successful at diagnosing most injected faults on all workloads, achieving high TP rates and low FP rates. We candidly discuss deviations where and why Kahuna-BB performed less than satisfactorily.

**PacketLoss:** The *PacketLoss1* problem was generally not diagnosed on all workloads, as compared to other problems, because Hadoop uses TCP, which provides some resilience against minor packet losses. The diagnosis of more severe packet losses had high TP rates, indicating we could detect the problem. However, the high FP rates indicated that we regularly indicted wrong nodes, due to the correlated nature of the problem since a packet loss on one node (e.g. due to a flaky NIC) can also register as a problem on other nodes communicating with it. Also, the *PacketLoss* problem was less successfully detected on `RandWriter` because its jobs largely involved disk I/O but minimal network communication.

**HANG-2080, HANG-1152:** The HANG-2080 and HANG-1152 problems affect the `Reduce` stage of computation. Since the `RandWriter` workload has no `Reduce` tasks, these hang problems have less impact on it (as shown in Table IV) than on `Sort` and `Pig`, which have relatively long `Reduces`. We could not diagnose the problem on the `Nutch` workload as it affected a majority of the nodes in the cluster, so that peer-comparison failed to diagnose this problem.

**DiskFull:** The *DiskFull* problem was not diagnosed successfully on the `Pig` workload, with relatively low TP rates. In this problem, the node with a full disk would use remote DataNodes rather than its local one to perform operations, so that workloads which perform more disk operations would be more greatly impacted. However, the `Pig` job in our experiments was largely compute-intensive, and less disk-intensive, so that the drop in disk activity on the problematic node did not cause the disk activity of that node to deviate significantly from those of other nodes.

## B. Kahuna-WB: White-box diagnosis

Kahuna-WB diagnosis was performed with the durations of each of the following states: `ReadBlock` and `WriteBlock` states on the DataNode, and the `Map`, `Reduce`, `ReduceCopy` and `ReduceMergeCopy` states on the TaskTracker. We found that diagnosis using the `Map` state was most effective; we summarize the results for diagnosis using all other states due to space constraints. The `Map` state was more effective for diagnosis as our candidate workloads spent the majority of

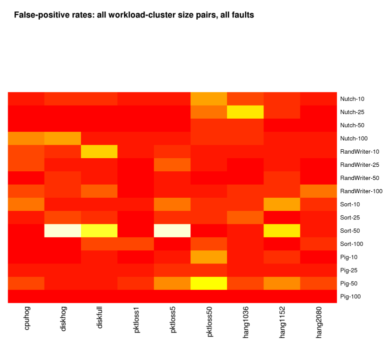

Fig. 8. False-positive rates for all problem-workload pairs (`Maps`).

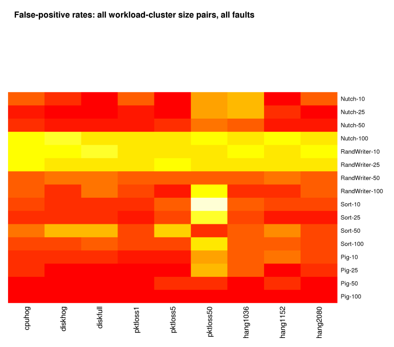

Fig. 10. False-positive rates for all problem-workload pairs (all other states).

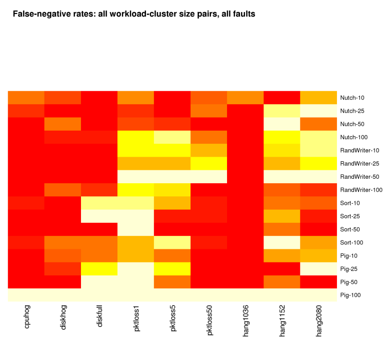

Fig. 9. False-negative rates for all problem-workload pairs (`Maps`).

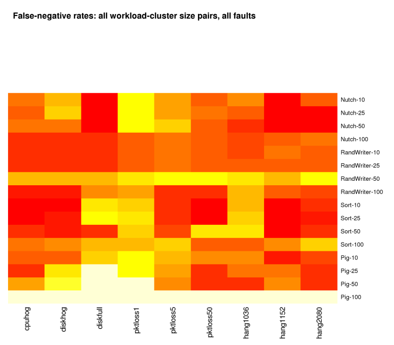

Fig. 11. False-negative rates for all problem-workload pairs (all other states).

their time in the `Map` state, while only `Sort` and `Pig` had significant `Reduce` tasks, so that the `Map` state was more likely to reflect problems. The `ReadBlock` and `WriteBlock` states were more likely to reflect problems in read-intensive and write-intensive workloads respectively, but were less effective on workloads with balanced I/O. The `ReduceCopy` and `ReduceMergeCopy` states were generally short-lived and occurred sparingly, and so were less effective. We present the FN ($= 1 - TP$) and FP rates for diagnosis using the `Map` state in figures 8 and 9 respectively, and the average FN and FP rates for diagnosis using all other states in figures 11 and 10 respectively. Each cell in the heatmap shows the FN or FP rate respectively for the given { problem, workload } pair for a given cluster-size (indicated after each workload). Deeper shades and deeper reds show low FN/FP rates, while lighter shades show high FN/FP rates.

**Diagnosis using `Maps`.** We diagnosed all problems relatively effectively, with low FN rates. However, we suffered high FP rates on the Packet Loss problems due to the correlated nature of packet losses, as explained in Section VIII-A. We also suffered high FP rates on HANG-1152 and HANG-2080 on `Nutch` and `RandWriter`, as explained in Section VIII-A.

**Diagnosis using other states** Non-`Map` states were less effective for diagnosis, achieving poorer (higher) FP and FN rates. In particular, they were ineffective for `RandWriter` because the workload had no `Reduce` and minimal `ReadBlock` states– this small number of non-`Map` states resulted in higher sensitivity of Kahuna-WB and hence higher FP rates.

### C. Discussion

Our experimental results show that our *peer-comparison* approach works well at diagnosing resource faults that result from over- or under-utilization of resources; e.g. in the CPU and DiskHog faults, compute and I/O capacities were excessively used, while in the HANG-1036 fault, compute capacities were underutilized due to the hang. This is reinforced by our diagnosis being more effective on faults that result in more severely increased job runtimes, in comparing Table IV with our diagnostic results. However, the *peer-comparison* approach works less well on communication failures e.g. packet losses. Also, Kahuna-WB does marginally better than Kahuna-BB at the more subtle faults, i.e. HANG-1152 and HANG-2080 in workloads with significant `Reduce` activity. Thus, our *peer-comparison* approach can be applied at different levels of granularity to enable more comprehensive diagnosis coverage.

## IX. Related Work

Diagnosing faults in distributed systems involves: (i) collecting system data, (ii) localizing faults to requests or nodes, and (iii) identifying root-causes. We compare our approach with recent work in some of these tasks. Also, we target MapReduce environments, which have long-lived jobs relative to Internet services with short-lived jobs and easily-defined SLOs that much recent work has dealt with [18], [4], [19], [6], [20].

**Instrumentation sources.** Magpie [18] uses black-box OS-level metrics, similarly to Kahuna-BB but at a finer granularity with request attribution, which Barham et al. achieve by exploiting expert input. Pip [21], X-trace [22] and Pinpoint [5] extract white-box metrics about paths through systems by tagging messages between components, while Kahuna-WB uses information extracted transparently from logs. [6] infers request paths by capturing messaging layer messages.

**Root-cause analysis.** Given failed requests, [4], [19] perform root-cause analysis on known-failed requests using clustering and decision trees respectively. [5], [20], [6] identify components along request paths, or anomalous paths, that contribute to failures/slowdowns. They uncover root-causes of known failed requests, while Kahuna-WB and Kahuna-BB identify requests with performance problems in systems where knowledge of SLO violations is not easily obtained.

**Peer-comparison techniques.** Peer comparison has been used to identify anomalous nodes or components, in actively replicated systems [24] and load-balanced web-service clusters [25], and from past views of peers in the system [20]. Slave nodes in MapReduce and Hadoop clusters are not strictly peers, as they handle different inputs; thus, it is interesting for our hypothesis of peer-symmetry to be borne out. [26] described initial work towards peer comparison in Dryad [27]. PeerPressure [28] diagnoses known configuration faults on a single host offline by comparing the suspect configuration with other labelled ones.

**Diagnosis in MapReduce and Hadoop.** [29] mined error messages DataNode logs to identify those correlated with actual errors, while we target performance problems. [22] was used to instrument Hadoop to return causal paths for identifying slow paths. [9] presented initial results in comparing state durations in Hadoop's execution, and [8] compared OS-level metrics; we evaluate these two techniques as part of the peer-similarity hypothesis extensively.

## X. Conclusion

Kahuna aims to diagnose performance in MapReduce systems based on the hypothesis that nodes exhibit *peer-similarity* under fault-free conditions, and that some faults result in *peer-dissimilarity*. This gives rise to our *peer-comparison* approach, applied to Hadoop, which we have validated using empirical evidence from real-world traces on the Yahoo! M45 cluster, and using experimental evaluation of two earlier peer-comparison algorithms on extensive benchmark and production workloads. Also, the applicability of our *peer-comparison* approach to multiple levels of Hadoop behavior suggests that this is a generally applicable approach, and our results show that applying *peer-comparison* to multiple levels (black-box system-level and white-box application-specific) of behavior can enhance the diagnosis coverage.

## References

[1] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *OSDI*, San Francisco, CA, Dec 2004, pp. 137–150.

[2] Apache Software Foundation, "Hadoop," 2007, http://hadoop.apache.org/core.

[3] ——, "Powered by Hadoop," 2009, http://wiki.apache.org/hadoop/PoweredBy.

[4] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *SOSP*, Brighton, United Kingdom, Oct 2005, pp. 105–118.

[5] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *DSN*, Bethesda, MD, Jun 2002.

[6] M. K. Aguilera, J. C. Mogul, J. L. Wiener, P. Reynolds, and A. Muthitacharoen, "Performance debugging for distributed system of black boxes," in *SOSP*, Bolton Landing, NY, Oct 2003, pp. 74–89.

[7] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," *Communcations of the ACM*, vol. 51, no. 1, pp. 107–113, Jan 2008.

[8] X. Pan, J. Tan, S. Kavulya, R. Gandhi, and P. Narasimhan, "Ganesha: Black-Box Diagnosis of MapReduce Systems," in *Workshop on Hot Topics in Measurement & Modeling of Computer Systems (HotMetrics)*, Seattle, WA, Jun 2009.

[9] J. Tan, X. Pan, S. Kavulya, R. Gandhi, and P. Narasimhan, "SALSA: Analyzing Logs as State Machines," in *USENIX WASL*, San Diego, CA, Dec 2008.

[10] S. Ghemawat, H. Gobioff, and S. Leung, "The Google file system." in *SOSP*, Lake George, NY, Oct 2003, pp. 29 – 43.

[11] Apache Software Foundation, "Apache's JIRA issue tracker," 2006, https://issues.apache.org/jira.

[12] Amazon Web Services LLC, "Amazon Elastic Compute Cloud," 2009, http://aws.amazon.com/ec2/.

[13] Yahoo! Inc., "Yahoo! reaches for the stars with M45 supercomputing project," 2007, http://research.yahoo.com/node/1884.

[14] S. Godard, "SYSSTAT," 2008, http://pagesperso-orange.fr/sebastien.godard.

[15] D. M. Endres and J. E. Schindelin, "A new metric for probability distributions," *Information Theory, IEEE Transactions on*, vol. 49, no. 7, pp. 1858–1860, 2003.

[16] D. R. A. Dempster, N. Laird, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the Royal Statistical Society*, vol. 39, pp. 1,38, 1977.

[17] C. Olston and B. Reed and U. Srivastava and R. Kumar and A. Tomkins, "Pig Latin: A Not-So-Foreign Language for Data Processing," in *SIGMOD*, Vancouver, BC, Canada, Jun 2008.

[18] P. Barham, A. Donnelly, R. Isaacs, and R. Mortier, "Using Magpie for request extraction and workload modelling," in *OSDI*, San Francisco, CA, Dec 2004.

[19] M. Chen, A. X. Zheng, J. Lloyd, M. I. Jordan, and E. Brewer, "Failure diagnosis using decision trees," in *ICAC*, New York, NY, May 2004, pp. 36–43.

[20] E. Kiciman and A. Fox, "Detecting application-level failures in component-based internet services," *IEEE Trans. on Neural Networks: Special Issue on Adaptive Learning Systems in Communication Networks*, vol. 16, no. 5, pp. 1027– 1041, Sep 2005.

[21] P. Reynolds, C. Killian, J. Wiener, J. Mogul, M. Shah, and A. Vahdat, "Pip: Detecting the unexpected in distributed systems," in *NSDI*, San Jose, CA, May 2006.

[22] R. Fonseca, G. Porter, R. Katz, S. Shenker, and I. Stoica, "X-Trace: A pervasive network tracing framework," in *NSDI*, Cambridge, MA, Apr 2007.

[23] S. Pertet, R. Gandhi, and P. Narasimhan, "Fingerpointing Correlated Failures in Replicated Systems," in *SysML*, Cambridge, MA, Apr 2007.

[24] M. Munawar, M. Jiang, and P. Ward, "Monitoring Multi-tier Clustered Systems with Invariant Metrics Relationships," in *International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, Leipzig, Germany, May 2008.

[25] G. Cretu-Ciocarlie, M. Budiu, and M. Goldszmidt, "Hunting for Problems with Artemis," in *USENIX WASL*, San Diego, CA, Dec 2008.

[26] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly, "Dryad: Distributed Data-Parallel Programs from Sequential Building Blocks," in *EuroSys*, Lisbon, Portugal, Mar 2007.

[27] H. Wang, J. Platt, Y. Chen, R. Zhang, and Y. Wang, "Automatic Misconfiguration Troubleshooting with PeerPressure," in *OSDI*, San Francisco, CA, Dec 2004.

[28] W. Xu, L. Huang, A. Fox, D. Patterson, and M. Jordan, "Detecting large-scale system problems by mining console logs," in *SOSP*, Big Sky, MT, Oct 2009.