# Diagnosing Heterogeneous Hadoop Clusters

**Shekhar Gupta**[1], **Christian Fritz**[1], **Johan de Kleer**[1], **and Cees Witteveen**[2]

[1] *Palo Alto Research Center, Palo Alto, CA 94304, USA*
{*Shekhar.Gupta, cfritz, dekleer*}*@parc.com*
[2] *Delft University of Technology, Delft, The Netherlands*
*C.Witteveen@tudelft.nl*

## ABSTRACT

We present a data-driven approach for diagnosing performance issues in heterogeneous Hadoop clusters. Hadoop is a popular and extremely successful framework for horizontally scalable distributed computing over large data sets based on the MapReduce framework. In its current implementation, Hadoop assumes a homogeneous cluster of compute nodes. This assumption manifests in Hadoop's scheduling algorithms, but is also crucial to existing approaches for diagnosing performance issues, which rely on the peer similarity between nodes. It is desirable to enable efficient use of Hadoop on heterogeneous clusters as well as on virtual/cloud infrastructure, both of which violate the peer-similarity assumption. To this end, we have implemented and here present preliminary results of an approach for automatically diagnosing the health of nodes in the cluster, as well as the resource requirements of incoming MapReduce jobs. We show that the approach can be used to identify abnormally performing cluster nodes and to diagnose the kind of fault occurring on the node in terms of the system resource affected by the fault (e.g., CPU contention, disk I/O contention). We also describe our future plans for using this approach to increase the efficiency of Hadoop on heterogeneous and virtual clusters, with or without faults.

## 1 INTRODUCTION

Hadoop[1] is a popular and extremely successful framework for horizontally scalable distributed processing of large data sets. It is an open-source implementation of the Google filesystem (Ghemawat *et al.*, 2003), called HDFS in Hadoop, and Google's MapReduce framework (Dean and Ghemawat, 2008). HDFS is able to store tremendously large files across several machines and using MapReduce, such files can be processed in a distributed fashion, moving the computation to the data, rather than the other way round. An increasing number of so called "big data" applications, incl. social network analysis, genome sequencing, or fraud detection in financial transaction data, require horizontally scalable solutions, and have demonstrated the limits of existing relational databases and SQL querying approaches.

Even though the value of Hadoop is widely acknowledged, the technology itself is still in its infancy. One of the problems that remains unsolved in the general case is the diagnosis of faults and performance issues in the cluster. Another shortcoming is Hadoop's implicit assumption of a homogeneous cluster, i.e., the assumption that all servers in the cluster have the same hardware and software configuration. If this assumption is violated, performance can be extremely poor. Therefore, in practice, Hadoop requires a homogeneous cluster. Given the vast number of existing servers with varying specifications in many organizations, it would be desirable to enable running Hadoop on such heterogeneous clusters as well. In this paper we present work in progress towards solving these two problems with particular emphasis on the diagnosis. We will describe in the future work section how we believe the same insights gained during diagnosis can be used to increase overall productivity of Hadoop on heterogeneous clusters, even in the absence of faults.

In its current version, Hadoop possesses only rudimentary monitoring capabilities. The primary mechanism for detecting faults is by pinging each compute node at a regular frequency. A node is considered dead if it fails to reply the ping within a certain time period. While this mechanism can be used to detect certain kinds of hard faults present in the cluster, performance issues that do not cause a node to go down but only slow it down and may be intermittent remain undetected.

In order to deal with hard faults that do not affect nodes at the level of the operating system where pings are affected, but which happen in user space (e.g., out of memory exceptions), Hadoop kills exceptionally long running tasks and reschedules them on a different node. This is problematic in particular on heterogeneous clusters where the required time to complete a task can vary strongly between different machines.

[1] `http://hadoop.apache.org/`

Recently, (Tan *et al.*, 2010b) proposed a diagnosis approach that uses a simple *peer similarity* model to identify faulty nodes in a Hadoop cluster. The idea underlying their approach is that the same task should take approximately the same amount of time on each node in the cluster. By building statistical models of task completion times over the nodes of the cluster, the authors are able to identify outliers with fairly high accuracy. However, the peer similarly model is again making the assumption that the cluster is homogeneous and is not applicable to heterogeneous clusters as we will show in Section 2.2.

We propose a data-driven diagnosis approach for detecting performance problems that is applicable to heterogeneous Hadoop clusters. Our approach constructs a behavioral model of every node for different classes of jobs, with one class for each dimension in the configuration space of the nodes, e.g., CPU speed, disk I/O speed, amount of RAM. The intuitive role for these *base* jobs is that their completion time will largely depend on the availability of the respective resource. The behavioral model based on these jobs can then be used for diagnosis. The model is used to determine how long a certain new job should take on a given node that is to be diagnosed, given the class of the new job, and the time it took to run on a different node.

In this paper, we present preliminary results that only consider problems that cause CPU or disk I/O contention and hence lead to degraded performance of jobs that make use of these resources. In future work, we plan to integrate this diagnostic information into the Hadoop scheduler to increase the overall productivity of the cluster in the face of faults as well as heterogeneity among nodes. The diagnostic information will allow the scheduler to make better predictions on task durations on specific nodes and hence create tighter schedules.

In the next section we will review Hadoop in more detail and summarize relevant related work. We then define the problem and describe our approach including empirical results. Before we conclude we outline our plans for future work.

## 2 BACKGROUND AND RELATED WORK

### 2.1 Hadoop

Hadoop is an open-source platform for distributed computing that currently is the de-facto standard for storing and analyzing very large amounts of data. Hadoop comprises a storage solution called HDFS, and a framework for the distributed execution of computational tasks called MapReduce. Figure 1 depicts how Hadoop stores and processes data. A Hadoop cluster consists of one *NameNode* and many *DataNodes* (tens to thousands). When a data file is copied into the system, it is divided up into *blocks* of 64MB. Each block is stored on three or more DataNodes depending on the replication policy of the deployed Hadoop cluster (Figure 1(a)). Once the data is loaded, computational *jobs* can be executed over this data. New jobs are submitted to the NameNode (Figure 1(b)). The NameNode will schedule *map* and *reduce tasks* onto the DataNodes.

- A map task is to process one block and generate a result for this block which gets written back to HDFS. Hadoop will schedule one map task for each block of the data, and it will do so, generally speaking, by selecting one of the three DataNodes that is storing a copy of that block to avoid moving large amounts of data over the network.
- A reduce task takes all these intermediate results and combines them into one, final result that forms the output of the computation.

In order to take advantage of Hadoop, a programmer has to write his program in terms a Map and a Reduce function. These functions are agnostic of the block structure and the distributed nature of the execution but typically rather operate at the level of a record or a line of text: some small unit of which the data is comprised. For each such unit, the Map function typically only performs rather basic operations.

The canonical example of a MapReduce program is `WordCount`, a program that counts the number of occurrences of each word in a large corpus of text. The Map function of `WordCount` tokenizes one block of the text, and counts words locally, line by line. The Reduce function would then take these local counts and sum them up to get the global result.[2]

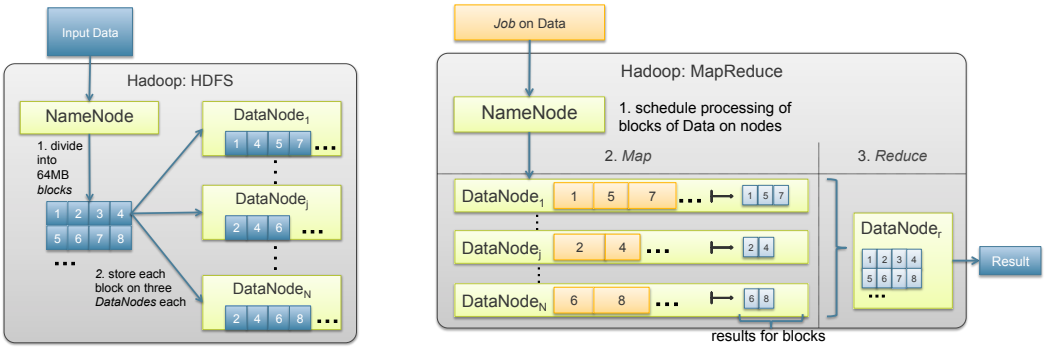
Since generally the Reduce Phase can only start once the Map Phase has been completed, the overall job completion time is typically determined by the time it takes the slowest DataNode to finish its assigned map tasks. Hadoop, as of the current version 0.22, does not take any performance differences between the DataNodes into account during the scheduling phase, but assumes a homogeneous cluster, i.e., servers that are equally fast in terms of CPU, disk I/O, RAM, and network bandwidth, etc.—the key-contributors to task completion time.

While Hadoop is particularly good for certain kinds of text analytics, in practice, various different kinds of jobs execute on a Hadoop cluster. For the purpose of this paper we will characterize jobs in terms of the types of resources they make use of most heavily. Many jobs are CPU intense, while others read from and write to disk a lot, or need to move larger amounts of data over the network. More specifically, we will generally characterize the map tasks of a job in this way and ignore the reduce tasks for the most part.

### 2.2 Related Work

Recently, (Tan *et al.*, 2010b) presented Kahuna, a diagnosis approach that uses a simple *peer similarity* model to identify faulty nodes in a Hadoop cluster. Roughly, the idea underlying their approach is that the same task should take approximately the same amount of time on each node in the cluster. More precisely, the authors build histograms of the time each task of a job takes on each machine. A node is identified as faulty when its histogram deviates from those of the other nodes. The authors show that this approach can detect slowdowns caused by various kinds of issues including CPU hogging, disk I/O hogging, as well as to a limited degree network package loss. The authors

[2] For parsimony we are omitting certain details in this example to the extend of which they are irrelevant for this paper.




(a) HDFS (b) MapReduce

Figure 1: HDFS stores files in blocks of 64MB, and replicates these blocks on three cluster nodes each. For a new *job* MapReduce then processes ("*maps*"') each block locally first, and then *reduces* all these partial results in a central location on the cluster to generate the end result.

also show that different workloads have different "diagnostic power" in the sense that certain issues are not uncovered by certain jobs. This is consistent with our assumption of different job classes. The authors do not describe whether Kahuna is able to detect what kind of fault may have occurred on a machine.

Kahuna assumes that the cluster is homogeneous, i.e., that tasks take roughly the same amount of time across machines. Figure 2 illustrates that, unsurprisingly, on heterogeneous clusters, the same task can take significantly longer or shorter depending on which machine is being used. During the experiment, we made sure that all machines were functioning flawlessly. Diagnosis hence cannot be based on the assumption that the same task should take equally long to execute on every node.

Many root cause analysis techniques use distributed monitoring tools that require active human intervention to locate the fault. Gangilia (Massie *et al.*, 2004) is a well known distributed monitoring system, which is capable of handling large clusters and grids. X-Trace (Fonseca *et al.*, 2007) and Pinpoint (Chen *et al.*, 2002) are tracing techniques to identify faults in distributed systems.

Tan *et al.* (2010a) developed a visualization tool to aid humans in debugging performance related issues of Hadoop cluster. The tool uses the log analysis technique SALSA (Tan *et al.*, ) that uses the Hadoop log files and visualizes a state-machine based view of every nodes' behavior. Ganesha (Pan *et al.*, 2008) is another diagnosis technique for Hadoop, which locates faults in MapReduce systems by exploiting OS level metrics. Konwniski and Zahari (2008) use X-Trace to instrument Hadoop systems to investigate Hadoop's behavior under different situations and tune its performance.

Automated performance diagnosis in service based cloud infrastructures is also possible via the identification of components (software/hardware) that are involved in a specific query response (Zhang *et al.*, 2007). The violation of a specified Service Level Agreement (SLA), i.e., expected response time, for one or more queries implies problems in one or more components involved in processing these queries. The methodology is widely accepted for many distributed systems. To a degree Hadoop is using this approach as well, as described above, but since the processing time for MapReduce jobs depend on many factors including the size of the data, only very crude limits can be used as cut-off. The determination of a reasonable cut-off is further hindered on heterogeneous clusters, where processing times can vary strongly between machines.

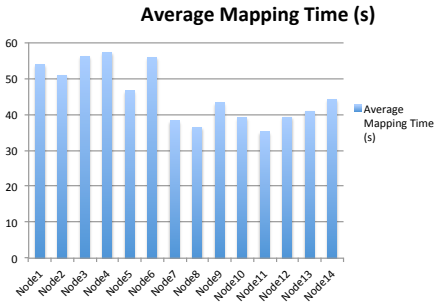


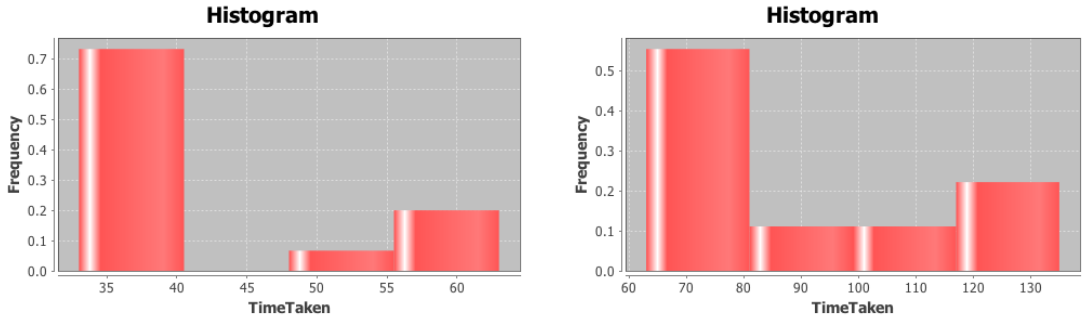

Figure 2: Average execution time of mapping tasks of the `WordCount` job on 14 heterogeneous nodes of the Hadoop cluster at PARC.

## 3 PROBLEM STATEMENT

Our primary goal in this work is to identify soft faults in heterogeneous Hadoop clusters, i.e., nodes that are slowed down, as well as to determine the cause for the degraded performance. Hadoop itself has a fault resilience component that continuously pings each DataNode to verify it is still alive. If a DataNode goes down or due to some other fault does not respond the ping, the NameNode marks the DataNode as dead and does not assign any more tasks to it. This mechanism is able to detect many hard faults in the cluster, but is unable to identify problems responsible for slowdowns.

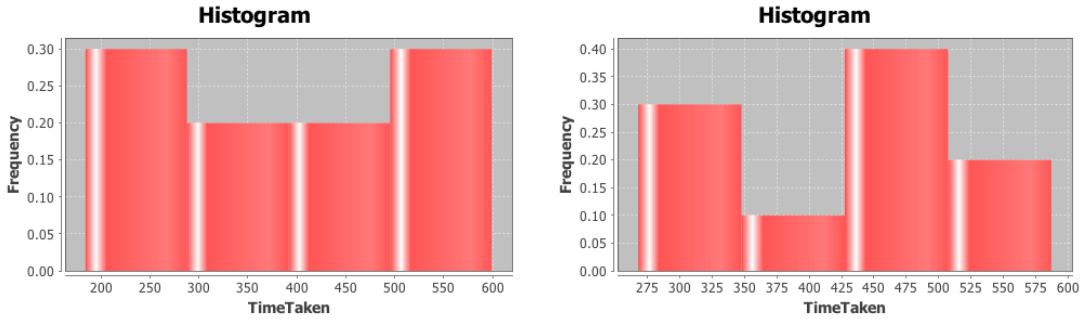
### Goals

The long term goal of this project is to enable the Hadoop scheduler to generate efficient schedules even




(a) Time distribution for `WordCount` without CPU hogging (b) Time distribution for `WordCount` **with** CPU hogging

Figure 3: CPU intensive MapReduce jobs such as `WordCount` are strongly affected in their completion time by over-subscription of the CPU (please note the number of seconds shown on the x-axis).




(a) Time distribution of `RandomWriter` without CPU hogging (b) Time distribution of `RandomWriter` **with** CPU hogging

Figure 4: MapReduce jobs that are not as CPU intensive such as `RandomWriter` are much less strongly affected in their completion time by over-subscription of the CPU.

on heterogeneous clusters and when some nodes are suffering from soft faults. This means that the scheduler will need to schedule fewer tasks on slower nodes, and for this it has to take the job characteristic resource profile into account as well as the performance characteristics of the nodes. As a pre-requisite of that, the focus of this paper is to determine the status of nodes in the cluster and learn both the job profiles as well as the relative node performance characteristics. The latter, intuitively, corresponds to a node's hardware configuration. When diagnosing a node to have a potential soft fault, we also want to determine the kind of fault. This can be accomplished by comparing the degree by which jobs with different resource profiles are affected by the fault.

### 3.1 Effect of Faults

In this paper we consider soft faults in the form of resource contention on a cluster node. For most resources there are numerous ways in which they can be oversubscribed, however, these root causes do not affect the performance in different ways, and we will only be able to determine which resources are affected, but not what is causing their contention.

For the purpose of validating our approach and in order to create the empirical results presented in the next section, we simulated two types of faults: CPU hogging, and disk I/O hogging. For CPU hogging, for instance, we ran additional programs on the node that used a lot of CPU time. As a result, tasks of CPU intensive jobs took much longer to complete, as shown in Figure 2.2 (please note the number of seconds on the x-axis). On the other hand, disk I/O intensive jobs such as the `RandomWriter` are not affected nearly as strongly, as can be seen in Figure 4. When the class of a job is known, this difference can be used to determine the likely type of soft-fault occurring on a node, i.e., the resource that is over-subscribed.

## 4 APPROACH

Our approach for determining performance problems of a heterogeneous Hadoop cluster intuitively consists of two steps. First we learn a model of every node in the cluster for each class of jobs. In the second step, these models are used to estimate how long a given new task should take on a given machine. Intuitively, if the new task takes much longer than predicted by the model, the node is likely to be suffering from a fault. By comparing the impact such a fault has on the completion time for tasks of different classes, i.e. different resource profiles, it is possible to diagnose the kind of

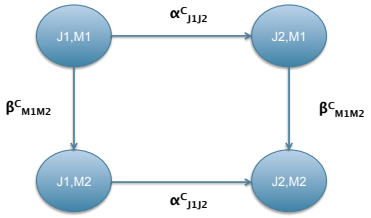



Figure 5: The *complexity coefficient* $\alpha$ describes the relative hardness or complexity of a job compared to other jobs and is assumed to be machine independent. Likewise, $\beta$ is a coefficient that captures the relative performance of one machine compared to another. It reflects the relative hardware configuration of the machines. Both of these coefficients are specific to a class of job, denoted $C$.

fault in the sense of determining which resource is affected (e.g., CPU, disk I/O).

### 4.1 Assumptions

Our approach makes a number of assumptions. For some of these assumptions we are presenting empirical evidence, others can be further relaxed in future work.

1. The resource profile of a given task is known.
2. For each job there is one resource that is the bottleneck which dominates the resource requirements for all map tasks of the job. For instance, a CPU intensive job is only marginally affected by disk I/O contention.
3. The class-specific, relative complexity of a task is machine independent. This means that for two jobs A and B of the same class, if B takes a factor of X longer than A on one node, then this factor will also apply to the completion times for A and B on another node, even if the two nodes have different configurations. We will further elaborate on this and provide empirical evidence for this assumption in Section 4.4.
4. The completion times for tasks executing on the same node are independent of each other.
5. The distribution of completion times for the tasks of a specific job on a specific node follow a Gaussian distribution. This means two things: repeating the exact same task multiple times would lead to a Gaussian distribution. But also the execution time for individual tasks of the same job, i.e., processing distinct 64MB blocks of the same data set using the same algorithm, follow this distribution.

   Our use of a Gaussian distribution in this paper is a first, pragmatic choice. In principle, a distribution that does not extend into the negative values would be a better choice for this application. Nevertheless, for the results presented in this paper, this misfit is fairly irrelevant.

Given these assumptions, our approach is captured by Figure 5: We can infer from the completion time of jobs J1 and J2 on machine M1 to the completion time of J2 on M2 once the completion time for J1 on M2 is known. This is possible based on Assumption 3.

### 4.2 Classes

Our diagnosis approach relies on an understanding of the resource profile of a task. Every task will make use of system resources in different ways. Some jobs are more CPU intensive, others read and write from and to disk more than others. In this paper, as stated in Assumption 2, we assume that for each job there is a single resource that forms the bottleneck. In this paper we only consider two classes: CPU intensive jobs and disk I/O intensive jobs. The assumption states that the intersection between these two classes is empty. In the rest of the paper, we will denote the class of jobs whose bottleneck is resource $R$ as $C_R$. Hence, we will be talking about two classes, $C_{CPU}$ and $C_{IO}$.

In our approach, we run our diagnosis for every class and in the end combine these diagnoses to recommend the specific root cause for a slowdown. For example, if we observe that on a particular node only CPU intensive jobs take longer than predicted by the model, this suggests that the node is suffering from CPU over-subscription, which could be caused, e.g., by some background, operating system, or zombie process.

#### Base Model of Class

In our approach we assume that initially, e.g., during first installation of the cluster, a set of prototypical, "*base*" jobs can be run on each machine. We assume that one such job for each *class* of jobs is run on each machine, where there is one class for each dimension in the configuration space, e.g., CPU speed, disk I/O speed, amount of RAM. This is discussed further in the next section.

For the two classes we are considering in this paper, $C_{CPU}$ and $C_{IO}$, `WordCount` and `RandomWriter` are selected as base jobs respectively. The model for another job of a class is determined relative to the base job model. Given job $J$ belonging to class $C_R$, we define $\alpha_J^{C_R}$, the *complexity coefficient* of job $J$ with respect to resource $R$, as the factor of how much longer $J$ takes to compute relative to the base job for $C_R$. Assumption 3 states that this factor is equal for every machine. That means that if, for instance, the original `WordCount` task on Node 1 takes 60 seconds, and another job that is also CPU intense takes 120 seconds on the same node, then we assume that the same factor of 2 would be observed on other nodes as well. Empirical justification and the procedure for estimating job complexity coefficients is described in Section 4.4.

### 4.3 Behavioral Model Construction

To learn the models we gather time samples of mapping tasks of every job from the log files generated by Hadoop. We collect the system logs produced by Hadoop's native logging feature on the NameNode. Subsequently, we parse these log files to collect timing samples for every node in the cluster. Each entry in the log can be treated as an event and is marked with a time stamp. An event is used to determine when a task is started and when it finishes on a specific node. The duration of mapping tasks is computed by subtracting those time stamps. We denote the average duration of all mapping tasks belonging to job $J$ on machine $M$ by $t_{J,M}$.

**Model Learning**
It is assumed that mapping task durations of the same job are distributed normally for any specific machine. The model for each job class $C_R$ is hence described by a mean $\mu_{J,R}$ and variance $\sigma_{J,R}$. For the base jobs of each class we can easily learn these parameters from the samples gathered during the initial execution of base jobs.

For each class $C_R$, we run the base job of the class on the Hadoop cluster and collect samples $t_{J,M,i}$ for every machine $M$. Recall that Hadoop schedules the execution of the tasks belonging to a job onto the available DataNodes according to where the data to be processed is stored. Since the data is distributed roughly uniformly, this process produces a number of samples for each machine. We assume that during this initial learning, no faults occur on the cluster.

For a job $J$, the mean and variance for machine $M$ can be estimated using the standard maximum likelihood estimator over the samples collected for tasks belonging to the base job of $C_R$:

$$\mu_{J,M} = \frac{1}{N_{J,M}} \sum_i t_{J,M,i} \tag{1}$$

$$\sigma^2_{J,M} = \frac{1}{N_{J,M}} \sum_i (t_{J,M,i} - \mu_{J,M})^2 \tag{2}$$

Where $N_{J,M}$ is the total number of samples collected for machine $M$ and job $J$. In our experiments, we use `WordCount` as the base job for class $C_{CPU}$, and `RandomWriter` for $C_{IO}$. We introduce the shorthand $J_{CPU} = WordCount$ and $J_{IO} = RandomWriter$ and will in the rest of the paper hence refer to $\mu_{J_{CPU},M}$ and $\sigma^2_{J_{CPU},M}$ as the model parameters of the base job for $C_{CPU}$ and $\mu_{J_{IO},M}$ and $\sigma^2_{J_{IO},M}$ respectively for $C_{IO}$.

### 4.4 Estimating Job Complexity

Assumption 3 states that the relative complexity of a job compared to its base job does not depend on the machine it is executed on. To verify this assumption we conducted an experiment with two new MapReduce jobs $J'_{CPU}$ and $J'_{IO}$. $J'_{CPU}$ belongs to class $C_{CPU}$ and $J'_{IO}$ belongs to $C_{IO}$. For every machine $M$ in the Hadoop cluster, we computed $\alpha^{CPU}_{J'_{CPU},M}$ and $\alpha^{IO}_{J'_{IO},M}$ as:

$$\alpha^{CPU}_{J'_{CPU},M} = \frac{\mu_{J'_{CPU},M}}{\mu_{CPU,M}} \tag{3}$$

$$\alpha^{IO}_{J'_{IO},M} = \frac{\mu_{J'_{IO},M}}{\mu_{IO,M}} \tag{4}$$

Figure 6 shows the results. The figure shows that the complexity coefficient for each class is fairly similar across machines, providing empirical justification for Assumption 3.

### 4.5 Diagnosis

Given the duration of mapping tasks belonging to a previously unseen job $J2$ on a number of machines $\{M_i\}_i$, we want to determine whether any of the nodes is having a fault and what the nature of the fault is. We do this by estimating for each machine how long each task should take under normal circumstances, and then use this information to determine a likelihood for a new observation (task duration) to indicate abnormality of the node. The idea for predicting the duration is that we can estimate the model, i.e., distribution of task duration, for a new task executing on a specific machine by scaling the base model of the class of the job for this machine by the complexity coefficient. The complexity coefficient in turn can be estimated from all other machines that have already run tasks of this job.

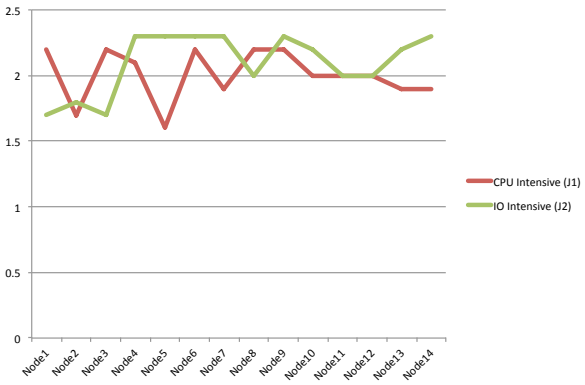



Figure 6: The values for complexity coefficient $\alpha$ for jobs belonging to $C_{CPU}$ (`WordCount`) and $C_{IO}$ (`RandomWriter`). Note that even though the nodes are heterogenous the relative complexity of a job is comparable between nodes.

**Predicting Job Completion Time**
When diagnosing machine $M_k$ for a job of class $C_R$, to get a better estimate of the complexity coefficient, every machine's $\alpha$ value for this job will be used except $M_k$'s. Hence, $\alpha^R_{J,*}$ can be estimated as

$$\alpha^R_{J,*} = \frac{1}{N-1} \sum_{i \neq k} \alpha^R_{J,M_i} \tag{5}$$

Where $N$ is the number of machines in the cluster that ran mapping tasks for this job.

On node $M_k$, the model for job $J2$ can be inferred from the model for $M_k$ for jobs of the respective class. Given the estimated complexity coefficient $\alpha^R_{J,*}$ for job $J2$ belonging to class $C_R$, the mean and variance for $J2$ on $M_k$ can be estimated as:

$$\mu^*_{J2,M_k} = \alpha^R_{J2,*} \cdot \mu_{R,M_k} \tag{6}$$

$$\sigma^*_{J2,M_k} = \alpha^R_{J2,*} \cdot \sigma_{R,M_k} \tag{7}$$

Recall that the approximate model of a job on a machine, characterized by these two parameters, describes the (approximate) distribution of durations of tasks of this job in this node. Therefore, the approximate model can be used for diagnosis, by computing the *relative likelihood* for an observed duration $x$ of a task, using the probability density function (PDF) of the underlying distribution, i.e., in this case, the PDF of the normal distribution:

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} \tag{8}$$

If a machine is suffering from over-subscription of a resource that is made intense use of by the task, then this will slow down the task and make the observed task completion time (duration) less likely. Hence, we consider a node $M_k$ to potentially have an over-subscription of resource $R$, if for a job $J'_R$ of class $C_R$, $f(t_{J'_R,M_k}; \mu^*_{J'_R,M_k}, \sigma^*_{J'_R,M_k})$ is significantly less than the average relative likelihood for the durations observed for tasks of $J'_R$ on all other machines. That is when:

$$f(t_{J'_R,M_k}; \mu^*_{J'_R,M_k}, \sigma^*_{J'_R,M_k}) \ll \frac{1}{N-1} \sum_{i \neq k} f(t_{J'_R,M_i}; \mu^*_{J'_R,M_i}, \sigma^*_{J'_R,M_i})$$

### 4.6 Experimental Evaluation

To evaluate the effectiveness of our diagnosis approach we collected task durations from the log files of our 14-node Hadoop cluster for two different instances of the `WordCount` and `RandomWriter` jobs. The 14 nodes are quite heterogeneous in their hardware configuration since these machines were purchased at different times and for different original purposes. This heterogeneity is reflected in Figure 2.

Following the described approach, we divided the experiments into two phases, a learning phase and a diagnosis phase. In the learning phase, we learned the model for the two base jobs, one for `WordCount` and one for `RandomWriter`, for each machine in the cluster. During this phase, we made sure that every node was functioning flawlessly.

During the diagnosis phase, we then simulated two types of resource contention: disk I/O contention on Node 10 and CPU contention on Node 13. These faults were simulated by running additional programs on the nodes that made excessive use of these resources ("hogging"). For CPU hogging we over-subscribed each CPU core on the node by a factor of two by running one extra program per core that ran an infinite loop with a basic arithmetic operation inside. The overall load per CPU core was hence around 2.0 when the Hadoop jobs were executing. For disk hogging, we ran a program that repeatedly wrote large files to the disks used by HDFS.

With these fault simulations in place, we ran a different `WordCount` job and a different `RandomWriter` job. These jobs differed from the jobs used as base jobs in that they repeated certain sub-tasks multiple times. This was to ensure these jobs have roughly the same resource profile as the base jobs but at the same time take noticeably longer. As can be seen in Figure 6, these jobs took roughly twice as long as their respective base job.

#### Results

Figure 7 summarizes the results of our experiment. On the x-axis we show the node number. For each node, two relative likelihood values for the observed average task durations are shown: one for the new `WordCount` job and one for the new `RandomWriter` job.

As can be seen, the relative likelihood for the observed task durations of the second `WordCount` job

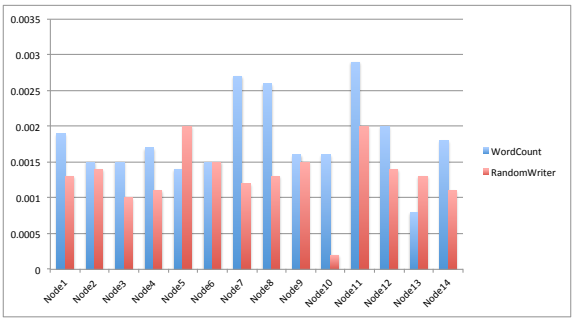



Figure 7: Results of the diagnosis for an instance of the `WordCount` and an instance of the `RandomWriter` job. On Node 10 we injected disk I/O contention, and on Node 13 we injected CPU contention. The y-axis shows the relative likelihood for observed task durations.

on Node 13 is significantly lower than the relative likelihoods for all other machines. Similarly, the relative likelihood for the observed task durations of the second `RandomWriter` job on Node 10 is significantly lower than the others. This suggests that it is indeed possible to use the presented approach to identify abnormally behaving nodes in a heterogeneous Hadoop cluster: Any node whose average task duration has a relative likelihood that is significantly lower than the average likelihood over other nodes is a candidate diagnosis. This is because, according to the model, it is unlikely to observe such average task durations on a node that is behaving normally. Applied to the results shown in the figure, this approach would correctly identify Nodes 10 and 13 as being abnormal.

Furthermore, note that the performance of Node 13 on the `RandomWriter` is not noticeably reduced by the presence of CPU contention, and that likewise the effect of disk I/O contention on Node 10 does not impact the completion time of the CPU intense `WordCount` job to a noticeable degree. This leads us to believe that this approach is indeed capable of determining the type of fault occurring on a node, at least to the level of detail of which resource is affected by the fault. Applied to our results, this approach would correctly determine that Node 10 is suffering from disk I/O contention, and Node 13 is experiencing CPU contention.

## 5 FUTURE WORK

The work presented in this paper is work in progress and the results are preliminary. This section outlines our plans for future work in this area. The longer term objective with this project is to increase productivity of Hadoop on heterogeneous and virtual clusters and in the face of faults, without human intervention. The key ideas for achieving this are as follows:

**Simultaneous machine status and job characterization.** We believe that it is possible to simultaneously learn about the status of nodes in the cluster and the resource profile of jobs, given a model of nominal behavior of nodes on canonical test jobs. While in this paper we were assuming that jobs belong to exactly one job class, in practice it is common for jobs to re-

quire a mix of resources. This is something we believe can be learned as well.

**Model transfer.** As depicted in Figure 5, we believe that it is possible to predict the time it takes a node to process a certain task, given the node's measured performance on an earlier (set of) job(s) and task durations on other machines. Likewise, it is possible to infer from one machine to another, even when the machines are equipped differently, by considering their models trained on prototypical jobs. We already explained and exploited this idea in this paper.

**Adaptive scheduling.** Given these improved predictions on how long a task should take on a specific node, it is possible to increase cluster productivity, in particular for (re-)scheduling of tasks on heterogeneous servers. Again, the reduce phase can typically only start once all mapping tasks have been completed. Therefore, a good prediction on how long a certain task will take on a certain node is critical to job completion time: without good predictions it is not possible to generate a schedule that is near-optimal.

**Pervasive diagnosis.** The two key ideas behind *pervasive diagnosis* (Kuhn *et al.*, 2008) are that performance measures of production jobs contain information about the health of the system, and that by directing the execution of jobs in a particular way, a system's state can be diagnosed without interrupting production by deriving insights from these performance measures. Even in the absence of faults, a Hadoop cluster, like many systems, has some degree of hidden state, i.e., aspects that are relevant but not directly observable. But since this hidden state affects job and task completion time, the hidden state can be diagnosed over time by deriving insights from these performance measures. Furthermore, scheduling a task of a known class (e.g., a CPU intensive job) onto a specific node can be used to "probe" the node and reveal the status of a specific system resource.

Applying these ideas, in future work, we plan to extend the existing Hadoop scheduling algorithms to proactively reduce the uncertainty about

(a) a node's health status in terms of its resources (CPU, RAM, disk I/O, network I/O, etc.), as well as the uncertainty about

(b) the resource requirements of previously unseen jobs.

Again, the purpose of reducing this uncertainty is to increase resilience of the system as well as reduce average job completion time via improved scheduling.

## 6 CONCLUSIONS

We have presented an approach for diagnosing performance issues in heterogeneous Hadoop clusters. The approach extends existing diagnosis approaches for Hadoop to clusters where the peer-similarity assumption does not hold, and further is able to distinguish between different types of faults. While the approach is based on a number of assumptions, we have presented empirical evidence for some of these assumptions. Further validating and/or relaxing some of the other assumptions is part of future work.

To empirically validate our diagnosis approach, we simulated soft faults in a Hadoop cluster consisting of 14 nodes, running two different MapReduce jobs with different resource profiles. The preliminary results presented in this paper suggest that the proposed approach is viable and able to achieve the intended goals of a) identifying machines on which resource contention is occurring, and b) determining the resource which is over-subscribed by considering the relative impact of faults on the completion time for jobs with different resource requirements.

**Acknowledgments**
We thank Bob Price, Rui Zhang, and the anonymous reviewers for their thoughts and constructive criticism.

## REFERENCES

(Chen *et al.*, 2002) Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, O Fox, and Eric Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *In Proc. 2002 Intl. Conf. on Dependable Systems and Networks*, pages 595–604, 2002.

(Dean and Ghemawat, 2008) Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, January 2008.

(Fonseca *et al.*, 2007) Rodrigo Fonseca, George Porter, Randy H. Katz, Scott Shenker, and Ion Stoica. X-trace: A pervasive network tracing framework. In *In NSDI*, 2007.

(Ghemawat *et al.*, 2003) Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The google file system, 2003.

(Konwniski and Zahari, 2008) Andy Konwniski and Matei Zahari. Finding the elephent in the data center: Tracing hadoop. Technical report, 2008.

(Kuhn *et al.*, 2008) Lukas Kuhn, Bob Price, Johan de Kleer, Minh Binh Do, and Rong Zhou. Pervasive diagnosis: The integration of diagnostic goals into production plans. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 1306–1312, 2008.

(Massie *et al.*, 2004) M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation, and Experience. *Parallel Computing*, 30(7), July 2004.

(Pan *et al.*, 2008) Xinghao Pan, Jiaqi Tan, Soila Kavulya, Rajeev G, and Priya Narasimhan. Ganesha: Black-box fault diagnosis for mapreduce systems. Technical report, 2008.

(Tan *et al.*, ) Jiaqi Tan, Xinghao Pan, Soila Kavulya, Rajeev G, and Priya Narasimhan. Salsa: Analyzing logs as state machines 1.

(Tan *et al.*, 2010a) Jiaqi Tan, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Visual, log-based causal tracing for performance debugging of mapreduce systems. In *ICDCS*, pages 795–806, 2010.

(Tan *et al.*, 2010b) Jiaqi Tan, Xinghao Pan, Eugene Marinelli, Soila Kavulya, Rajeev Gandhi, and Priya Narasimhan. Kahuna: Problem diagnosis for mareduce-based cloud computing environments. In *IEEE/IFIP Network Operations and Management Symposium*, 2010.

(Zhang *et al.*, 2007) Rui Zhang, Steve Moyle, Steve McKeever, and Alan Bivens. Performance problem localization in self-healing, service-oriented systems using bayesian networks. In *Proceedings of the 2007 ACM symposium on Applied computing*, SAC '07. ACM, 2007.