Challenges to Error Diagnosis in Hadoop Ecosystems

Jim (Zhanwen) Li[1], Siyuan He[2], Liming Zhu[1,3],Xiwei Xu[1],
Min Fu[3], Len Bass[1,3], Anna liu[1,3], An Binh Tran[3]
[1]NICTA, Sydney, Australia
[2]Citibank, Toronto, Canada
[3]School of Computer Science and Engineering, University of New South Wales, Sydney, Australia

Abstract

Deploying a large-scale distributed ecosystem such as HBase/Hadoop in the cloud is complicated and error-prone. Multiple layers of largely independently evolving software are deployed across distributed nodes on third party infrastructures. In addition to software incompatibility and typical misconfiguration within each layer, many subtle and hard to diagnose errors happen due to misconfigurations across layers and nodes. These errors are difficult to diagnose because of scattered log management and lack of ecosystem-awareness in many diagnosis tools and processes.

We report on some failure experiences in a real world deployment of HBase/Hadoop and propose some initial ideas for better trouble-shooting during deployment. We identify the following types of subtle errors and the corresponding challenges in trouble-shooting: 1) dealing with inconsistency among distributed logs, 2) distinguishing useful information from noisy logging, and 3) probabilistic determination of root causes.

1. Introduction

With the maturing of cloud and Hadoop technologies, more and more organizations are deploying and using systems in the Hadoop ecosystem for various purposes. Hadoop is an ecosystem that consists of multiple layers of largely independently evolving software and its deployment is across distributed nodes and different layers.

Even for experienced operational professionals with limited experience with the Hadoop ecosystem, the deployment and use is highly error-prone and error diagnosis and root cause identification takes a significant amount of time.

Traditionally, logs and error messages are important sources of information for error diagnosis. In a distributed system, logs are generated from multiple sources with different granularities and different syntax and semantics. Sophisticated techniques have been proposed to produce better logs or analyze existing logs to improve error diagnosis. However, there are a number of limitations of the existing approaches for the situation outlined above.

Consider one of the error messages that we encountered in our experiments "java.net.ConnectException: Connection refused ". One existing approach is to correlate error messages with source code. Yet knowing where in the Java library this message was generated will not help determine the root cause. The cause of this error is, most likely, a misconfiguration but it is a misconfiguration that indicates inconsistency between multiple items in the ecosystem. Trouble shooting an error message such as this requires familiarity with the elements of the ecosystem and how they interact. This familiarity is primarily gained through experience, often painful. Furthermore, the messages leading up to this error message may be inconsistent or irrelevant. They are usually voluminous, however.

Providing assistance to non-expert installers of a complicated eco-system such as HBase/Hadoop is the goal of the work we report on here. In this paper, we report some failure experiences in real world deployments of HBase/Hadoop. Specifically, we focus on three key challenges: 1) dealing with inconsistency among distributed logs, 2) distinguishing useful information from noisy logging, and 3) probabilistic determination of root causes.

There are two assumptions about this work. First, it came out of observing and studying errors committed by non-expert installers of Hadoop ecosystems. Our target is system administrators and non-experts in HBase/Hadoop. Second, we assume that the developers of such systems will not change the way they record logs significantly although we do hope they produce them with operators more in mind. Thus our initial solutions are around dealing with inconsistency and uncertainties with existing logs. The case studies are all based on an Hadoop/HBase [3][5] cluster running on AWS EC2s[2].

The contributions of this paper include:

1. Identification of different types of errors in Hadoop ecosystem deployment using real world cases and investigations into the root causes of these errors. The majority of errors can be classified into four types:

- **Operational errors** such as missing/incorrect operations and missing artifacts. Errors introduced during restarting/shutting down nodes, artifacts (files and directories) not created, created with the wrong permission or mistakenly moved and disallowed operations due to inconsistent security environment are the major ones.

- **Configuration errors** include errors such as illegal, lexical, and syntax errors in standalone software systems and cross-systems/nodes inconsistency in an ecosystem.
- **Software errors** include compatibility issues among different parts of an ecosystem (e.g. HBase and HDFS compatibility issues) and bugs.
- **Resource errors** include resource unavailability or resource exhaustion, especially in cloud environment, that manifest themselves in highly uncertain ways and lead to system failures.

The diagnosis of these errors and locating the true causes is more difficult in an ecosystem setting, which leads to our second contribution.

2. Identified specific error diagnosis challenges in multi-layer ecosystems deployed in distributed systems: 1) dealing with inconsistency among distributed logs, 2) distinguishing useful information from noisy logging, and 3) probabilistic determination of root causes. These highlighted the gaps in the current approaches and lead to our third contribution.

3. Introduced a new two-phase error diagnosis general framework for distributed software ecosystem from the operator (rather than the developer) perspective. This new approach attempts to remove some inconsistency and noise by combining phase-one local diagnosis with phase-two global diagnosis and produces a probability-ranked list of potential root causes. This simplifies the complexities of constructing correlations between logging information and root causes.

2. Related Works

In previous work, efforts have been placed into the improvement of logging mechanisms for providing more comprehensive system information to assist system management. For example, Apache Flume [2] aims to offer a scalable service for efficiently collecting, aggregating, and moving large amounts of log data in large-scale distributed computing environments. Similar logging systems include Facebook Scribe [9], Netflix Edda [13] and Chukwa [16], which are systems for aggregating real-time streams of log data from a large number of servers. These developments of logging systems provide a good basis for collecting up-to-date system information in complex distributed systems, but they do not have the capability to bridge the gap between logging information and error diagnosis.

Another focus of research of using logging information to assist troubleshooting is to explore effective machine learning approaches for mining critical messages associated with known problems. For example, Xu et. al. [21] studied the correlation between logs and source code. In [12], Nagaraj et. al. troubleshoot performance problems by using machine learning to compare system logging behaviors to infer associations between components and performance. In [11], Narasimhan and her team members studied the correlation of OS metrics for failure detection in distributed systems. In [24][25][26], Zhou's research group studied the trace of logging information in source codes, and introduced a new logging mechanism to locate the position of bugs with more efficiency. And in [15], Oliner et. al. studied the connections between heterogeneous logs and quantified the interaction between components using these logs. There is a general lack of ecosystem awareness in these tools and the ability to deal with log inconsistency and uncertainty as well as cross system incompatability.

Misconfigurations are another significant issues leading to software system errors. Zhou and her colleagues conducted an empirical study over different types of misconfigurations and their effects on systems by studying several open source projects, including MySQL, Tomcat and etc. [23]. They focus on the misconfigurations of each individual system, while the correlation of configurations across systems, especially in a distributed environment, is ignored. Randy Katz and his colleagues [17] studied the connection between configuration and software source code to improve misconfiguration detection but did not cover the connection between configurations and logs, which is critical to operators.

These existing works give a good basis for understanding some challenges in error diagnosis. But many studies are from the viewpoint of software developers rather than operators. They also did not consider issues around the connections among the logs and configurations at different layers and across different nodes.

3. Case Study: HBase Cluster on Amazon EC2

Our case study comes from a real world privacy research project where the goal is to process large amounts of anonymised information using different approaches to see if one can still infer identity from the information. Several sub-projects want to share a HBase/Hadoop cluster which is deployed in Amazon EC2. The operators and users of the cluster are IT-savvy researchers and system admins but not Hadoop or distributed system experts. Although Amazon provides an Elastic Map Reduce (EMR) system with Hadoop pre-installed, the different requirements of the sub-projects led to a fresh deployment on EC2 virtual machines.

An HBase/Hadoop cluster consists of Hadoop Distributed File System (HDFS) for distributed files storage, Zookeeper for distributed service coordination, and HBase for fast individual record lookups and updates in distributed files. Each node in an HBase cluster consists of multiple layers of software systems, shown as Figure 1 (a). Every layer must perform in a correct manner to ensure the communication across layers/nodes and overall system availability, as shown in Figure 1 (b).

The communication between nodes in a Hadoop ecosystem relies on SSH connections, so security, ports and protocols required by SSH must be available. Hadoop, Zookeeper and HBase rely on Java SDK. Updated versions of Java that are compatible are necessary. The Hadoop layer is the basis of an HBase cluster. This layer is controlled by HDFS and MapReduce [3]. The configurations over the Namenode and all Datanodes [3] must be correct, ensuring the communication and computation over this layer, so that clients of Hadoop can access HDFS or MapReduce services. (HBase does not need MapReduce, but applications of HBase may require MapReduce). Zookeeper performs a role of distributed service coordinator for HBase. Its responsibilities include tracking server failures and network partitions. Without Zookeeper, HBase is not operational. Based on these underlying distributed services, HBase requires communication between the HMaster and the Regional Servers [5] in the HBase layer. The full deployment and running of some of our small programs went through several false starts in a matter of weeks by different people independently. We asked the people to record their major errors, diagnosis experiences and root causes.

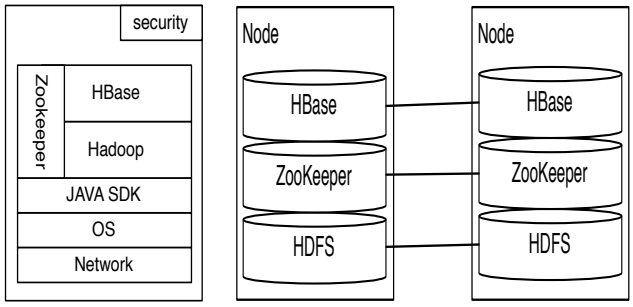

Figure 1 Layers of software systems in Hadoop

4. Logging Exceptions and Uncertainties in Determining Root Causes

In Table 1, we list some key examples of logs and error messages collected in our Hadoop/HBase deployment process. The "Logging Exception" column records the error messages when the deployment process got interrupted. The "Possible Causes" column listed the possible causes and the relevant information that different operators mentally considered or physically examined during error diagnosis. For errors that are related to connection issues, we use Src and Dest to respectively represent the source and destination nodes.

Table 1: Logging Exceptions and Potential Causes

	Source	Logging Exception	Possible Causes: Required Information for Examination
1	HBase/Hadoop	"org.apache.hadoop.hdfs.server.datanode.DataNode: DataNode is shutting down: org.apache.hadoop.ipc.RemoteException: org.apache.hadoop.hdfs.protocol.UnregisteredDatanodeException"	In the problematic DataNodes: • Instance is down: *ping, ssh connection* • Access permission: *check authentication keys, check ssh connection* • HDFS configuration: *conf/slaves* • HDFS missing components: *check the datanode setting and directories in hdfs*
2	Zookeeper	"java.net.UnknownHostException at org.apache.zookeeper.ZooKeeper.<init>(ZooKeeper.java:445)"	In Src and Dest nodes: • DSN: *DSN configuration and testing* • Network connection: *ssh testing* • Zookeeper connection: *JPS and logging messages in zoo.out* • Zookeeper configuration: *zoo.cfg* • Zookeeper status: *processes (PID and JPS)* • Cross-node configuration consistency
3	HDFS/ MapReduce / HBase/ Zookeeper	"java.net.ConnectException: Connection refused "	In Src and Dest: • Network connection: *ping IPs, ping hostnames and check ssh connection* • Security setting: *check ssh connection* and *check authentication keys* • Hostname/IP/Ports configuration: *check configuration files, netstat and lsof* • Software status: *check processes* • Software compatibility: *detect and check system and library versions* • Cross-layer configuration consistency • Cross-node configuration consistency
4	HBase/Hadoop	"org.apache.hadoop.hdfs.server.namenode.NameNode: java.lang.IllegalArgumentException: Does not contain a valid host:port authority: file"	In Src and Dest: • Missing configuration files: *hostfile, hadoop configuraitons* • Security file missing or incorrect: *connection permission, host/port permission* • Host and Port setting in HDFS: *core-site.xml, hdfs-site.xml* • Host and Port settings in DNS • Network host and port settings: *netstat, lsof etc* • Cross-node configuration consistency
5	HBase/Hadoop	"org.apache.hadoop.hdfs.server.common.InconsistentFSStateException: Directory	In the problematic nodes: • Missing files in HDFS file system: *look for directory in hdfs*

		/app/hadoop/tmp/dfs/name is in an inconsistent state: storage directory does not exist or is not accessible."	• Missing/Incorrect operations on HDFS: *hdfs format* • Directory misconfiguration: *core-site.xml*
6	HBase/Hadoop	"WARNorg.apache.hadoop.metrics2.impl.MetricsSystemImpl: Source name ugi already exists! ERRORorg.apache.hadoop.hdfs.server.datanode.DataNode: java.io.IOException: Incompatible namespaceIDs in /app/hadoop/tmp/dfs/data:"	In the problematic NameNode and DataNode: • Misconfigurations on the hadoop: *scan the name space setting in hadoop* • File System duplication: *scan the hdfs file system* • Other nodes with the same name started: *scan configurations and hostfiles*
7	Zookeeper	"JMX enabled by default Using config: /home/ubuntu/zookeeper-3.4.5/bin/../conf/zoo.cfg Error contacting service. It is probably not running."	In the problematic Nodes: • Misconfigurations on JAVA: *Java version and Java Path* • Missing components in JAVA: *Update Java version* • JAVA configurations in Zookeeper: *JAVA_HOME Path* • Zookeeper configurations: configurations in zoo.cfg • Zookeeper version problem: *the compatibility of Zookeeper, JAVA and OS*
8	Hadoop/MapReduce	In deployment testing, "class is not found: maxtempreturemapper , and the job is not defined in the jar , when running map reduce jobs…"	In the problematic Nodes: • Misconfiguration in Jobtracker: *the path to the MapReduce Jar* • Misconfigurations in MapReduce: *mapred-site.xml* • Class compiling by JAVA: *the Java compiler* • The correctness of the Jar file: *the source code of the MR application*
9	HBase/Hadoop	"ERROR org.apache.hadoop.security.UserGroupInformation: PriviledgedActionException as:ubuntu cause:java.io.IOException: File /app/hadoop/tmp/mapred/system/jobtracker.info could only be replicated to 0 nodes, instead of 1"	In the problematic Nodes: • Security setting: *RSA settings* • Directory configuration: *scan configuration files core-site.xml* • HDFS files system directories: *scan the Hadoop file system, run hadoop scripts, or scan hadoop log*
10	HBase/Hadoop	"FATAL org.apache.hadoop.hdfs.StateChange: BLOCK* NameSystem.getDatanode … ERROR org.apache.hadoop.security.UserGroupInformation: PriviledgedActionException as:ubuntu cause:org.apache.hadoop.hdfs.protocol.UnregisteredDatanodeException"	In the problematic Nodes: • Hadoop misconfiguration: *scan hadoop configuration files core-site.xml and conf/slaves* • HDFS not formatted: *scan hadoop file system, run hadoop scripts* • HBase Configurations: *scan HBase configurations conf/hbase-site.xml* • Cross-layer configuration consistency: *scan the configurations with dependencies in HBase and Hadoop* • System security: *test SSH conncetions*
11	HBase/Hadoop	"org.apache.hadoop.hbase.client.RetriesExhaustedException: Failed setting up proxy interface	In the Src and Dest Nodes: • Hadoop status: *scan processes by PID and JPS, use Hadoop commands* • Hadoop client and server configurations: *the master name setting in hdfs* • Permission in the system: *RSA and ssh connnections* • Cross-layer configuration consistency: *HBase configurations is inconsistent to the Hadoop configuraitons, e.g., the ports and the names of file systems*
12	HBass/Hadoop	"WARN org.apache.hadoop.hdfs.server.datanode.DataNode: java.io.IOException: Too many open files at java.io.UnixFileSystem.createFileExclusively(Native Method) at java.io.File.createNewFile(File.java:883) …	In nodes used by HBase • configuration of HBase: *maximum number of files setting* • Workload of HBase: *under heavy work load* • Configuration of Hadoop: *maximum number of files setting* • OS environment misconfiguration: e.g. *default ulimit (user file limit) on most unix systems insufficient*
13	Hadoop	"org.apache.hadoop.hdfs.DFSClient: DataStreamer Exception: org.apache.hadoop.ipc.RemoteException: java.io.IOException: File /app/hadoop/tmp/mapred/system/jobtracker.info could only be replicated to 0 nodes, instead of 3"	In Src and Dest Nodes • Hadoop Status: *scan processes by PID and JPS, use Hadoop commands* • MapReduce Status: *scan processes by PID and JPS, use MapReduce commands* • Directory in Hadoop *configurations: the number of replicas in hdfs-site.xml, the number of slaves in conf/slaves* • *Connection problems: e.g. node IP configurations* • HDFS file system: *the directory does not exist in the HDFS* • Cross-node configuration consistency: the Hadoop states in each node
14	Zookeeper	"org.apache.zookeeper.ClientCnxn: Session 0x23d41f532090005 for server null, unexpected error, closing socket connection and attempting reconnect"	In Src and Dest Nodes • Zookeeper Configurations: *the clinet port, name of nodes etc. in zoo.cfg* • Network Configurations: *the ssh connections to other nodes* • Security Configurations: *the RSA settings* • Cross-node configuration consistency: *the zookeeper configurations in each node, the configuration over networks in each node* • States of Zookeeper: *running, waiting or failed*
15	HBase/Hadoop/Zookeeper	"FATAL org.apache.hadoop.hbase.regionserver.HRegionServer: ABORTING region server hbaseSlave1,60020,1362958856599: Unexpected exception during initialization, aborting org.apache.zookeeper.KeeperException$ConnectionLossException: KeeperErrorCode =	In Src and Dest Nodes • HBase configurations: *the zookeeper setting in HBase, conf/hbase-site and conf/hbase-env.sh*, *the authority to use Zookeeper from HBase* • The OS/Network problem on the nodes: *the ssh connection and the compatibility between JAVA, HBase and OS* • Zookeeper configurations: *the Zookeeper availability* • Cross-layer configuration consistency: *the ports, quorum and authority setup*

		ConnectionLoss for /hbase/master at org.apache.zookeeper.KeeperException.create(KeeperException.java:99)"	*in zookeeper and HBase*

From the operator experiences in the project, locating a root cause from a logging exception is very difficult. A logging exception could result from multiple causes while the connections to these causes are not obvious from an error message. For example, a logging "java.net.ConnectException: Connection refused", shown in Figure 2, has at least 10 possible causes. And exceptions on different software (in the ecosystem) or on different nodes are sometimes inconsistent but related in a direct and indirect manner. It is an extremely exhausting search process to locate a root cause in a large-scale domain with highly coupled information and many uncertainties.

In this study, we classify the error analysis into three layers: exception, source and cause. Exception is the error message returned in log files or console; source is defined as the component that originally leads to this exception message; and cause is the reason that the source got the exception. And we classify errors into four groups: operations, configurations, software and resources. We use these classifications in our proposed approach to organize local diagnosis and a global diagnosis.

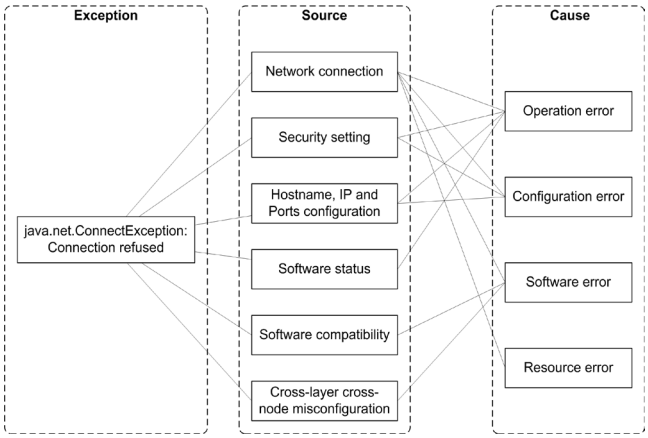

Figure 2 Three layers of error diagnosis: exception-source-cause

Configuration errors

Misconfigurations include legal ones with unintended effects and illegal ones (e.g. lexical, and syntax errors) that are commonly seen in standalone software systems [23]. We also include the cross-domain inconsistent configurations in such distributed ecosystems. The later one is more difficult to detect because all configurations must be taken as a whole for error examination. We give an example that caused issues in the project.

Example 1. HDFS directory used in HBase must be consistent with the Hadoop file system default name. In HBase, hbase-site.xml, the setting of hbase.rootdir:

```
<property>
    <name>hbase.rootdir</name>
    <value>hdfs://hbaseMaster:54310/hbase</value>
</property>
```

must be consistent with the setting of fs.default.name in Hadoop core-site.xml

```
<property>
    <name>fs.default.name</name>
    <value>hdfs://hbaseMaster: 54310/value>
</property>
```

Mismatch of these configurations results in failures of HBase startup. For an enterprise HBase cluster deployment, such as CDH4, there are hundreds of options requiring customizable configurations in 20+ sub-systems [7][17]. These configurations are inter-correlated, but misconfigurations are hard to detect.

Operation errors:

Operation errors include missing operations and incorrect operations. Operation errors cause missing components and abnormal system behaviors, resulting in software failures. For example, HDFS initialization requires a newly formatted file system. Inconsistent File System State Exception shown below will return if this required operation was missing. The formatting is performed externally. The message is not obviously interpretable to lack of formatting.

Example 2:

```
org.apache.hadoop.hdfs.server.common.InconsistentFSStateException: Directory /app/hadoop/tmp/dfs/name is in an inconsistent state: storage directory does not exist or is not accessible.
```

Software errors

Software errors came from software incompatibility and bugs. One instance is the incompatibility between Hadoop 0.20.x version and HBase 0.90.2, resulting in potential data loss [14]. Another commonly seen failure due to system incompatibility is certain required Java libraries do not exist. Such case usually happens because of the incompatibility between Java and the OS, and so some required Java libraries are not installed. Here are two examples of logging errors returned by Hadoop and Zookeeper installation in the project. However, both messages are not at all clear about the root causes and can lead operators to the wrong places. But after examining related logs in other layers, the root cause was located.

Example 3: JAVA problem in Hadoop:

Error msg: "java[13417:1203] Unable to load realm info from SCDynamicStore" when running any HDFS command

Example 4: JAVA problem in ZooKeeper:

JMX enabled by default Using config: /home/ubuntu/zookeeper3.4.5/bin/../conf/zoo.cfg Error contacting service. It is probably not running.

Resource errors

Resource errors refer to resource unavailability occurring in the computing environment. For example, limitation of disk I/O (or failure of SAN disks) could result in significant performance degradation in some nodes, resulting in some exceptions of *timeout*. However, one key challenge is that many such resource errors are hidden in log files and not correlated with respective resource metrics. Only by looking at different logs from different layers of software in the ecosystem, can the root cause be identified.

5. Discussion: Three Challenges to Troubleshoot Errors with Logs

Logs guide error diagnosis. There are three challenges that should be addressed for achieving more accurate and efficient error diagnosis in distributed ecosystem.

5.1 Dealing with inconsistency among logs

Inconsistent loggings around states and events introduce significant issues to error diagnosis. Inconsistency may occur in a single log file, across multiple log files in different components. Inconsistency of logging information includes two types: inconsistent contexts and inconsistent timestamps.

Taking a Hadoop ecosystem as an example, an ecosystem consists of a large number of interacting heterogeneous components. Each component has logging mechanism for capturing specific states and events, what messages are put into log files is often determined by the requirements of component itself with no global coordinator for managing these logging messages across components. The decisions of what states and events are put into the log file under what context are not the same in different components. When taking these logging messages across components as a whole for error diagnosis, missing, redundant and contradictory information may introduce context inconsistency.

Another type of inconsistency comes from inconsistent timestamps in large-scale systems where network latency cannot be ignored. Information logging could be asynchronous as errors and other corresponding information are written into log files. This asynchronous logging contributes to risks of timing inconsistency, which may be misleading in error diagnosis and omit correlated events. Solutions to timing correlation problems exist such as NTP[1] and Google Spanner [8] but these solutions are not currently implemented in our test stack. Again, we are attempting to deal with what is, rather than what should be.

5.2 Distinguishing useful information from noisy logging

Large-scale distributed systems are constantly producing a huge amount of logs for both developers and operators. Collecting all of them into a central system is often itself a significant challenges. Systems have emerged to create such centralized log collection, for example Scribe from Facebook, Flume from Apache, Logstash[2] and Chukwa [16] .

Due to the large amount of information available, error diagnosis is often very time-consuming whether it is done by humans querying the centralized log system or through machine learning systems across all the logs. Traditional error analysis algorithms could encounter scalability issues dealing with a large number of logging messages. Some scalable clusters for logging analysis were developed for addressing this issue [21][22]. But these solutions focus on offline analysis to identify source code bugs while operation issues often require online or nearline analysis putting significant challenge to the analysis infrastructure and algorithm. Thus, it is important to discard noise earlier and effectively for different types of errors at different times.

In many cases, such as performance issues and connection problems, additional tests and associated logs are required for analysis. They are often time consuming if planned and done reactively through human operators. These additional tests should be incorporated into the error diagnosis tools and logging infrastructure so they are automatically carried out at certain stage of the error diagnosis or proactively done, adding more useful signals to the error diagnosis process.

5.3 Probabilistic determination of root causes dealing with uncertain correlations

In error diagnosis, correlation of logging events is critical for identifying the root causes. Many machine-learning techniques have been developed for exploring the correlated events in log files in order to construct more accurate and more comprehensive models for

[1] http://en.wikipedia.org/wiki/Network_Time_Protocol

[2] http://logstash.net/

troubleshooting [11]. However, uncertainties in logs introduce significant challenges in determining root causes Uncertainties in log files are often caused by missing logging messages, inconsistent information and ambiguity of logging language (lexical and syntax). We classify the uncertainties into four types:

Uncertainties Between Exceptions
In distributed systems, an error occurring in one place often triggers a sequence of responses across a number of connected components. These responses may or may not introduce further exceptions at different components. However, simply mining exception messages from these distributed log files may not detect the connections among these exceptions. Known communications between components should be considered in correlating exceptions and comparing different root causes diagnosis at each component or node.

Uncertainties Between Component States
Accurate logging states and context help filter useless information and guides error diagnosis. They are important information for understanding component statuses and limiting the scope for searching the root cause to errors. Logging states could be fully coupled or fully independent, or with somehow indirect connections. But these dependent relationships among state logging are not described in log files. And missing and inconsistent states logging may further introduce uncertainties in the relationships between states. Dependencies in an ecosystem must be taken into consideration when analysing state logs.

Uncertainties Between Events
In error diagnosis exploring the coherence of logging events is a critical task for tracking the change of system subject to errors, providing a basis for inferring the root cause from exceptions. A challenge for constructing event coherence is uncertainties lying in the relationships between logging events. These uncertainties destroy connections between information, losing data for modeling the sequence of system change subject to errors.

Uncertainties Between States And Events
In most cases, logging states and events must be considered at the same time for modeling the system behavior in terms of logging conditions. Ideally logging messages deliver details of events and of corresponding sates across this process. But this obviously is over optimistic. In most log files the connections between states and events contain uncertainties, which destroy the event-state mapping, creating a gap for finding the root causes from logging errors.

6. A Two-Phase Error Diagnosis Framework

The above challenges are the consequence of current logging mechanisms and overall designs, which are often out of the control of the users. So error diagnosis requires an effective approach that is capable of figuring out the most possible root causes for errors despite of the inconsistency, noise and uncertainty in logs. To achieve this goal in a large-scale distributed computing environment, we are working on two ideas. The first idea is to treat the operations as a set of explicit processes interacting with each other. We model and analyze these processes and track the their progression at runtime. We use the processes to connect seemingly independent events and states scattered in various logs and introduce "process context" for error diagnosis [27]. In this paper, we introduce the second idea, which proposes a two-phase error diagnosis framework for error diagnosis. The first-phase error diagnosis is conducted at each distributed node with agents for local troubleshooting, and a second-phase is performed on a centralized server for global error diagnosis to compare the various local diagnoses and deal with node-to-node errors. Unlike existing solutions that have a centralized database aggregating all logging information, in our approach information is highly filtered for the second-phase diagnosis depending on the error types, environment and local diagnosis.

A framework of this design is shown in Figure 3. The key is to let each node or log-file propose a set of potential causes for the errors (if there are logging exceptions in the file) and gather the states of the relevant components, then send these likely causes and component states to a centralized second-phase diagnosis for probability-ranked list of causes using a gossip algorithm [19]. The logging information that we consider in this framework includes log files from software components, e.g. Hadoop, Zookeeper and HBase, and historical information of resource components, which include records of resource (CPU/Memory) consumption, disk I/O, network throughput, and process states monitored by agent-based systems (e.g. JMX and Nagios in our environment). All of these are seen as log files of components in our approach.

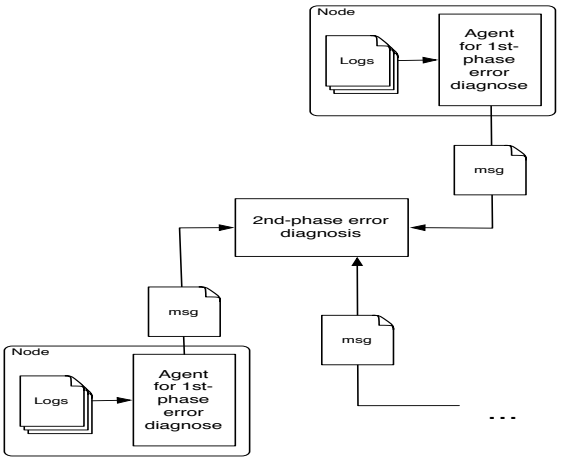

Figure 3: Architecture of the 2-phase error diagnosis

6.1 The first-phase error diagnosis

The first-phase error diagnosis is conducted with agents located at each distributed node for identifying the errors in the components in the node. This process is described with Figure 4.

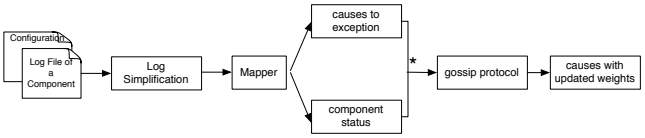

Figure 4: The process of error diagnosis in the first-phase

Inputs to an agent include log files of components and configuration files. An agent first summarizes each log file, which is a process to convert logging information into a standard format with consistent terms (lexical and syntax) for later identification. This operation is conducted in the stage of log simplification. For each summarized log file given by the log simplification, the agent uses a mapper, which is a small expert knowledge base responsible to deliver a set of likely causes in response to the logging exception. A mapper offers: a) a list of candidate causes that may contribute to the logging exceptions (which include ERROR and WARNING messages), denoted by C_e^r, standing for a cause r that may lead to an exception e in component C. Each cause may include a set of sub-causes $C_{e'}^{r'}$, and b) the status of the component, denoted by C_s, which includes the status of domain name, ports, accounts, security, tractable actions for software components, and utilization and performance for resource components. Each C_e^r is associated with a weight w, whose initial value is 1. We define a tuple $[C_e^r, w]$ to indicate this relationship. These proposed causes and monitored component statuses are considered as a whole by a gossip algorithm, updating the weight w of each C_e^r with a rule: when a cause C_e^r conflicts to a component status C_s', the associate weight w is reduced by 1; and when a cause C_e^r is supported by another log file, the weight w is then increased by 1. This strategy reduces the number of correlated features (across logging messages) that are less related to errors, creating potential for handling complex problems in a large-scale systems.

6.1.1 An Example

The following is an example for troubleshooting cross-system inconsistent configuration within an HBase cluster. Cross-system misconfiguration is hard to detect because it is difficult to trace exceptions across multiple systems. In an HBase node (with IP: 10.141.133.22, which is a master node in this cluster), it includes log files respectively from HBase, Hadoop, Zookeeper. When a log file from HBase returns an exception, shown as:

> 2013-03-08 09:31:44,934 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: hbaseMaster/10.141.133.22:9000. Already tried 9 time(s).
>
> 2013-03-08 09:31:44,938 FATAL org.apache.hadoop.hbase.master.HMaster: Unhandled exception. Starting shutdown.
>
> java.net.ConnectException: Call to hbaseMaster/10.141.133.22:9000 failed on connection exception: java.net.ConnectException: Connection refused
>
> …
>
> ERROR org.apache.hadoop.hbase.master.HMasterCommandLine: Failed to start master

,where hbaseMaster is a domain name defined in the configuration file. In the phase-one error diagnosis, this logging information is summarized as:

> HBaseMaster:9000 failed
> connection exception: hbaseMaster/10.141.133.22:9000

The mapper takes this as input and returns a set of likely causes to the exception and gives the states of the component:

> 1. Hadoop server: unavailable ($C_{e\_HD}^{avail}$: 1)
> 2. Hadoop server: not accessible ($C_{e\_HD}^{access}$:1), whose prerequisite: $C_{s\_HD}^{avail}$=*true*, and with sub causes:
>   a. domain name is unavailable ($C_{e\_HD}^{hbaseMaster:9000}$:1)
>   b. user account is unavailable ($C_{e\_HD}^{act}$:1)
>   c. security setting is unavailable ($C_{e\_HD}^{sec}$:1)
>
> status: HBase HMaster: failed ($C_{s\_HBseHMaster}^{avail}$ = *false*).

In the same node, the log file from Hadoop NameNode gives the information

> 2013-03-08 09:23:02,362 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: Roll FSImage from 10.141.133.22
>
> 2013-03-08 09:23:02,362 INFO org.apache.hadoop.hdfs.server.namenode.FSNamesystem: Number of transactions: 0 Total time for transactions(ms): 0Number of transactions batched in Syncs: 0 Number of syncs: 1 SyncTimes(ms): 8

Because there are no logging exceptions, no causes are proposed from this Hadoop log file. So it can give: $C_{s\_HD}^{avail}$=*true*. And since no configurations regarding account and security are found in the configuration files, it gives $C_{s\_HD}^{act}$ =*true* and $C_{s\_HD}^{sec}$ =*true*. And it can be achieved from the summary of Hadoop log that: $C_{s\_HD}^{hbaseMaste:54310}$=*true*, where *hbaseMaste:54310* is the domain name of Hadoop.

A combination of this information given by Mappers is used in the Gossip protocol for updating the weight associated with each proposed cause. Output is shown as below. The reasons are described in the bracket in the right-hand side.

> 1. [$C_{e\_HD}^{avail}$: 0] ($C_{e\_HD}^{avail}$ =*false* conflicts $C_{s\_HD}^{avail}$=*true*)
> 2. [$C_{e\_HD}^{access}$: 1] ($C_{s\_HD}^{avail}$=*true*, and no information directly related to $C_{e\_HD}^{access}$)
>   a.[$C_{e\_HD}^{hbaseMaster:9000}$: **1**] (no information directly related to

$C_{e\_HD}^{hbaseMaster:9000}$)
b.[$C_{e\_HD}^{act}$:0] ($C_{e\_HD}^{act}$ =*false* conflicts $C_{s\_HD}^{act}$=*true*)
c.[$C_{e\_HD}^{sec}$:0] ($C_{e\_HD}^{sec}$ =*false* conflicts $C_{s\_HD}^{sec}$=*true*)

The cause to this "java.net.ConnectException" is limited to the availability of domain name of *hbaseMaster:9000* (cause 2.a). Although this approach does not provide a 100% accurate error diagnosis, it shows the possibility of using limited information to sort out the most likely causes for a logging error in a complex computing environment with many connected systems.

6.2 The second-phase error diagnosis

The second-phase error diagnosis offers troubleshooting for the exceptions that may be across multiple nodes. This process sorts out the possibility of causes that are delivered by the agents in the first-phase error diagnosis.

Each agent summaries the output of the first-phase error diagnosis into a message, which includes the likely causes with updated weights (if the weight is greater than zero), and the status of each component.

6.2.1 An Example

For example, the agent in the above node will deliver the second-phases error diagnosis a message with the information of:

Agent ID: 10.141.133.22
HBase Log:
[$C_{e\_HD}^{access}$: 1]
[$C_{e\_HD}^{c}$:1]
$C_{s\_HBseHMaster}^{avail}$ = *false*, $C_{s\_HBseHMaster}^{act}$ =*true*, $C_{s\_HBseHMaster}^{sec}$ =*true*
Hadoop Log:
$C_{s\_HD}^{avail}$=*true*, $C_{s\_HD}^{hbaseMaste:54310}$=*true*, $C_{s\_HD}^{act}$ =*true*, $C_{s\_HD}^{sec}$=*true*
Zookeeper Log:
[$C_{e\_ZK}^{hbaseSlave3:3888}$:1]
$C_{s\_ZK}^{avail}$=*true*, $C_{s\_ZK}^{myID:1}$=*follower*, $C_{s\_ZK}^{act}$ =*true*, $C_{s\_ZK}^{sec}$=*true*

This message includes the information of Zookeeper. Because there is a WARN message given in the Zookeeper log file, shown as:

Cannot open channel to 4 at election address hbaseSlave3/10.151.97.82:3888

and the Zookeeper status has shown that this Zookeeper quorum is performing follower, a possible cause with weight is shown as [$C_{e\_ZK}^{hbaseSlave3:3888}$:1]. This warning error message is related to anther Zookeeper quorum on: *hbaseSlave3:3888*. It is handled by the second-phase error diagnosis.

For this troubleshooting, input information regarding this error for the second-phase error diagnosis includes:

Agent ID: 10.141.133.22, prose error [$C_{e\_ZK}^{hbaseSlave3:3888}$:1]

Agent ID: 10.36.33.18, where the zookeeper quorum is selected as leader, propose error [$C_{e\_ZK}^{hbaseSlave3:3888}$:1]
Agent ID: 10.151.97.82, where locate the problematic zookeeper quorum *hbaseSlave3:3888*

Because the Zookeeper status is found in the Agent ID: 10.151.97.82, the weight of $C_{e\_ZK}^{hbaseSlave3:3888}$ is updated to

[$C_{e\_ZK}^{hbaseSlave3:3888}$:**2**]

in the second-phase error diagnosis with the gossip protocol to find out the most likely cause to guide troubleshooting. It locates the issue on the zookeeper quorum on 10.151.97.82. And since no states of this Zookeeper quorum are returned, the focus of troubleshooting can be limited on:

Network communication between nodes, and
Configurations of the zookeeper quorum in Zookeeper and HBase

This simple example shows that the 2-phase error diagnosis can use existing limited information to determine a list of ranked possible causes to logging errors dealing with uncertainty challenges we identified earlier. And the strategy is simple to implement as it uses an existing gossip algorithm to compare local diagnosis, which could be in turn based on past work and ad-hoc knowledge database, and it can handle cross-layer and cross-node errors.

7. Conclusions and Future Works

Using a real world case study, we identified some difficult-to-diagnosis errors committed by non-expert Hadoop/HBase users. We classified errors and documented the difficulties in error diagnosis, which led to three key challenges in ecosystem error diagnosis. We proposed a simple and scalable two-phased error diagnosis framework that only communicates the absolute necessary information for global diagnosis after local diagnosis. We experimented and demonstrated the feasibility of the approach using a small set of common Hadoop ecosystem errors. We are currently implementing the full framework and performing large-scale experiments.

8. Acknowledgement

NICTA is funded by the Australian Government as represented by the Department of Broadband, Communications and the Digital Economy and the Australian Research Council through the ICT Centre of Excellence program.

9. References

[1] AWS EC2, http://aws.amazon.com/ec2/
[2] Apache Flume, http://flume.apache.org/ 2013
[3] Apache Hadoop, http://hadoop.apache.org/ 2013
[4] Apapche Hadoop, "HDFS High Availability Using the Quorum Journal Manager", 2013, http://hadoop.apache.org/docs/r2.0.3-alpha/hadoop-

yarn/hadoop-yarn-site/HDFSHighAvailabilityWithQJM.html
[5] Apache HBase, http://hbase.apache.org/ 2013
[6] Apache Zookeeper, http://zookeeper.apache.org/ 2013
[7] Cloudera "CDH4 Installation Guide" 2013 http://www.cloudera.com/content/cloudera-content/cloudera-docs/CDH4/latest/PDF/CDH4-Installation-Guide.pdf
[8] Corbett, J.C., et al. Spanner: Google's Globally-Distributed Database, Processings OSDI '12, Tenth Symposium on Operating System Design and Implementation, Hollywood, Ca, October, 2012.
[9] Facebook Scribe, https://github.com/facebook/scribe
[10] Lars George, "HBase: The Definitive Guide", *Publisher: O'Reilly Media*, 2011
[11] Soila P. Kavulya, Kaustubh Joshi, Felicita Di Giandomenico, Priya Narasimhan , "Failure Diagnosis of Complex Systems", In *Journal of Resilience Assessment and Evaluation of Computing Systems* 2012, pp 239-261
[12] Karthik Nagaraj, Charles Killian, and Jennifer Neville. "Structured Comparative Analysis of Systems Logs to Diagnose Performance Problems". In the *Proceedings of 9th USENIX Symposium on Networked Systems Design and Implementation* (NSDI '12). San Jose, CA. 25-27 April, 2012.
[13] Netflix Edda, https://github.com/Netflix/edda
[14] Michael G. Noll, "Building an Hadoop 0.20.x Version for HBase 0.90.2", 2011, http://www.michael-noll.com/blog/2011/04/14/building-an-hadoop-0-20-x-version-for-hbase-0-90-2/
[15] Adam J. Oliner , Ashutosh V. Kulkarni , Alex Aiken. "Using correlated surprise to infer shared influence", In the *Proceedings of Dependable Systems and Networks* (DSN), 2010, June 28 2010-July 1 2010,
[16] Ariel Rabkin, Randy Katz "Chukwa: a system for reliable large-scale log collection", in *Proceedings of the 24th international conference on Large installation system administration* (LISA 10), 2010
[17] Ariel Rabkin, "Using Program Analysis to Reduce Misconfiguration in Open Source Systems Software", Ph.D thesis 2012
[18] Patrick Reynolds, Charles Killian, Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah, and Amin Vahdat. "Pip: Detecting the Unexpected in Distributed Systems". In *proceedings of Networked Systems Design and Implementation* (NSDI 2006). May 2006
[19] Devavrat Shah, "Gossip Algorithm", MIT, 2009, http://web.mit.edu/devavrat/www/GossipBook.pdf
[20] Tom White, "Hadoop: The Defintitive Guide", the second edition ,published *by O'Reilly Media* 2010
[21] Wei Xu, "System Problem Detection by Mining Console Logs", Ph.D thesis, *EECS, UC Berkeley*, Aug. 2010
[22] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan, "Large-scale system problem detection by mining console logs", In *Proceeding of the 22nd ACM Symposium on Operating Systems Principles* (SOSP' 09), Big Sky, MT, October 2009
[23] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N. Bairavasundaram and Shankar Pasupathy. "An Empirical Study on Configuration Errors in Commercial and Open Source Systems." In *the Proceedings Of The 23rd ACM Symposium On Operating Systems Principles* (SOSP'11), October 2011
[24] Ding Yuan, Soyeon Park, and Yuanyuan Zhou. "Characterising Logging Practices in Open-Source Software". In *the Proceedings of the 34th International Conference on Software Engineering* (ICSE'12), Zurich, Switzerland, June 2012
[25] Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou and Stefan Savage. "Improving Software Diagnosability via Log Enhancement". In *ACM Transactions on Computer Systems* (TOCS), Vol. 30, No. 1, Article 4, Februray 2012.
[26] Ding Yuan, Soyeon Park, Peng Huang, Yang Liu, Michael M. Lee, Xiaoming Tang, Yuanyuan Zhou and Stefan Savage. "Be Conservative: Enhancing Failure Diagnosis with Proactive Logging" In the *Proceedings of the 9th ACM/USENIX Symposium on Operating Systems Design and Implementation* (OSDI'12), Hollywood, CA,
[27] X. Xu, L. Zhu, J. Li, L. Bass, Q. Lu, and M. Fu, "Modeling and Analysing Operation Processes for Dependability," in IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Fast Abstract, 2013