Detecting Large-Scale System Problems by Mining Console Logs

Author: Wei Xu, Ling Huang, Armando Fox, David Patterson, Michael Jordan
Conference: ICML 2010
SOSP2009, ICDM 2009

Reporter: Zhe Fu
Outline

• Introduction
• Key Insights
• Methodology
• Evaluation
• Online Detection
• Conclusion
Introduction

Background of console logs
• Console logs rarely help in large-scale datacenter services
• Logs are too large to examine manually and too unstructured to analyze automatically
• It’s difficult to write rules that pick out the most relevant sets of events for problem detection

Anomaly detection
• Unusual log messages often indicate the source of the problem
Introduction

Related work:
  • as a collection of English words
  • as a single sequence of repeating events

Contributions:
  • A general methodology for automated console log processing
  • Online problem detection with message sequences
  • System implementation and evaluation on real world systems.
Key Insights

Insight 1: Source code is the “schema” of logs.

- Logs are quite structured because generated entirely from a relatively small set of log printing statements in source code.

```java
starting: xact 325 is COMMITTING
starting: xact 346 is ABORTING

1  CLog.info("starting: " + txn);
2  Class Transaction {
3      public String toString() {
4          return "xact " + this.tid +
5              " is " + this.state;
6      }
7  }
```

- Our approach can accurately parse all possible log messages, even the ones rarely seen in actual logs.
Key Insights

Insight 2: Common log structures lead to useful features.

- Message types: marked by constant strings in a log message
- Message variables:
  - Identifiers: variables that identify an object manipulated by the program
  - State variables: labels that enumerate a set of possible states an object could have in program

```java
1    CLog.info("starting: " + txn);
2    Class Transaction {
3        public String toString() {
4            return "xact " + this.tid +
5                " is " + this.state;
6        }
7    }
```
Key Insights

Insight 2: Common log structures lead to useful features.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Examples</th>
<th>Distinct values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Identifiers</td>
<td>transaction_id in Darkstar; block_id in Hadoop FS; cache_key in Apache server; task_id in map-reduce.</td>
<td>many</td>
</tr>
<tr>
<td>State Vars</td>
<td>Transaction stages in Darkstar; Server names in Hadoop; HTTP status code (200, 404); POSIX process return values.</td>
<td>few</td>
</tr>
</tbody>
</table>
Key Insights

Insight 3: Message sequences are important in problem detection.

• Messages containing a certain file name are likely to be highly correlated because they are likely to come from logically related execution steps in the program.

• Many anomalies are only indicated by incomplete message sequences.

For example, if a write operation to a file fails silently (perhaps because the developers do not handle the error correctly), no single error message is likely to indicate the failure.
Key Insights

Insight 4: Logs contain strong patterns with lots of noise.

• normal patterns—whether in terms of frequent individual messages or frequent message sequences—are very obvious frequent pattern mining and Principal Component Analysis (PCA)

• Two most notable kinds of noise
  • random interleaving of messages from multiple threads or processes
  • inaccuracy of message ordering

grouping methods
Case Study

<table>
<thead>
<tr>
<th>System</th>
<th>Lang</th>
<th>Logger</th>
<th>Msg Construction</th>
<th>Lines of Code</th>
<th>Lines of Logs</th>
<th>Vars</th>
<th>Parse</th>
<th>ID</th>
<th>ST</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Linux (Ubuntu)</td>
<td>C</td>
<td>custom</td>
<td>printk + printf wrap</td>
<td>7477k</td>
<td>70817</td>
<td>70506</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Low level Linux services</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bootp</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>11k</td>
<td>322</td>
<td>220</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>DHCP server</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>23k</td>
<td>540</td>
<td>491</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>DHCP client</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>5k</td>
<td>230</td>
<td>205</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>ftpd</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>3k</td>
<td>66</td>
<td>67</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>openssh</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>124k</td>
<td>3341</td>
<td>3290</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>crowd</td>
<td>C</td>
<td>printf</td>
<td></td>
<td>7k</td>
<td>112</td>
<td>131</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Kerberos 5</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>44k</td>
<td>6261</td>
<td>4971</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>iptables</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>52k</td>
<td>2341</td>
<td>1528</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Samba 3</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>566k</td>
<td>8461</td>
<td>6843</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Internet service building blocks</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Apache 2</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>312k</td>
<td>4008</td>
<td>2835</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>mysql</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>714k</td>
<td>5092</td>
<td>5656</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>postgresql</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>740k</td>
<td>12389</td>
<td>7135</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Squid</td>
<td>C</td>
<td>custom</td>
<td>printf wrap</td>
<td>148k</td>
<td>2675</td>
<td>2740</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Jetty</td>
<td>Java</td>
<td>log4j</td>
<td>string concatenation</td>
<td>138k</td>
<td>699</td>
<td>667</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Lucene</td>
<td>Java</td>
<td>custom</td>
<td>custom log function</td>
<td>217k</td>
<td>143</td>
<td>159</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>BDB (Java)</td>
<td>Java</td>
<td>custom</td>
<td>custom trace</td>
<td>260k</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Web Applications</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MoinMoin</td>
<td>Python</td>
<td></td>
<td>string replacement</td>
<td>96k</td>
<td>566</td>
<td>611</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Trac</td>
<td>Python</td>
<td></td>
<td>string replacement</td>
<td>84k</td>
<td>104</td>
<td>65</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>AppEngine SDK</td>
<td>Python</td>
<td>custom</td>
<td>string replacement</td>
<td>122k</td>
<td>378</td>
<td>349</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Flash Game Client</td>
<td>ActionScript</td>
<td>AS3 built in</td>
<td>string concatenation</td>
<td>4k</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Distributed systems</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hadoop</td>
<td>Java</td>
<td>custom log4j</td>
<td>string concatenation</td>
<td>173k</td>
<td>911</td>
<td>1300</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Darkstar</td>
<td>Java</td>
<td>jdk-log</td>
<td>Java format string</td>
<td>90k</td>
<td>578</td>
<td>658</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Nutch</td>
<td>Java</td>
<td>log4j</td>
<td>string concatenation</td>
<td>64k</td>
<td>507</td>
<td>504</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Cassandra</td>
<td>Java</td>
<td>log4j</td>
<td>string concatenation</td>
<td>46k</td>
<td>393</td>
<td>437</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Storage Prototype</td>
<td>C</td>
<td>custom</td>
<td>custom trace</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Google System 1</td>
<td>C++</td>
<td>custom</td>
<td>string concatenation</td>
<td>-</td>
<td>10k</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Google System 2</td>
<td>C++</td>
<td>custom</td>
<td>string concatenation</td>
<td>-</td>
<td>21k</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Google System 3</td>
<td>C++</td>
<td>custom</td>
<td>string concatenation</td>
<td>-</td>
<td>6k</td>
<td>-</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>
Methodology

Step 1: Log parsing
• Convert a log message from unstructured text to a data structure

Step 2: Feature creation
• Constructing the state ratio vector and the message count vector features

Step 3: Machine learning
• Principal Component Analysis (PCA)-based anomaly detection method

Step 4: Visualization
• Decision tree
Step 1: Log parsing

message types  identifiers  state variables
starting: xact 325 is COMMITTING
starting: xact 346 is ABORTING

regular expression:
starting: xact (. *) is (. *)

• Challenge: Templatize automatically
  • C language
    • fprintf(LOG, "starting: xact %d is %s")
  • Java
    • CLog.info("starting: " + txn)

• Difficulty in OO (object-oriented) language
  • We need to know that CLog identifies a logger object
  • OO idiom for printing is for an object to implement a toString() method that returns a printable representation of itself for interpolation into a string
  • Actual toString() method used in a particular call might be defined in a subclass rather than the base class of the logger object
Step 1: Log parsing

Parsing Approach - Source Code

Figure 3: Using source code information to parse console logs.
Step 1: Log parsing

Parsing Approach - Source Code
Step 1: Log parsing

Parsing Approach - Logs

• Apache Lucene reverse index
• Implement as a Hadoop map-reduce job
  • Replicating the index to every node and partitioning
  • The map stage performs the reverse-index search
  • The reduce stage processing depends on the features to be constructed
Step 2: Feature creation

State ratio vector

- Each state ratio vector: a group of state variables in a time window
- Each vector dimension: a distinct state variable value
- Value of the dimension: how many times this state value appears in the time window

choose state variables that were reported at least \(0.2N\) times
choose a size that allows the variable to appear at least \(10D\) times in \(80\%\) of all the time windows
Step 2: Feature creation

**Message count vector**
- Each message count vector: group together messages with the same identifier values
- Each vector dimension: different message type
- Value of the dimension: how many messages of that type appear in the message group

<table>
<thead>
<tr>
<th>starting</th>
<th>prepare</th>
<th>committed</th>
</tr>
</thead>
<tbody>
<tr>
<td>xact 325</td>
<td>COMMITTING</td>
<td>xact 325</td>
</tr>
<tr>
<td>PREPARING</td>
<td>COMMITTED</td>
<td></td>
</tr>
<tr>
<td>325: 111000000</td>
<td>326: 101000000</td>
<td>327: 111010000</td>
</tr>
</tbody>
</table>

**Algorithm 1** Message count vector construction

1. Find all message variables reported in the log with the following properties:
   - Reported many times;
   - Has many distinct values;
   - Appears in multiple message types.
2. Group messages by values of the variables chosen above.
3. For each message group, create a message count vector $y = [y_1, y_2, \ldots, y_n]$, where $y_i$ is the number of appearances of messages of type $i$ ($i = 1 \ldots n$) in the message group.
Step 2: Feature creation

Message count vector
Step 2: Feature creation

**State ratio vector**
- Capture the aggregated behavior of the system over a time window

**Message count vector**
- Help detect problems related to individual operations

<table>
<thead>
<tr>
<th>Feature</th>
<th>Rows</th>
<th>Columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status ratio matrix $Y^s$</td>
<td>time window</td>
<td>state value</td>
</tr>
<tr>
<td>Message count matrix $Y^m$</td>
<td>identifier</td>
<td>message type</td>
</tr>
</tbody>
</table>

Table 4: Semantics of rows and columns of features

Also implement as a Hadoop map-reduce job
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

Principal Component Analysis (PCA) problem formulation

Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find $k$ vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

Andrew Ng
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

PCA is not linear regression
Step 3: Machine learning

Intuition behind PCA anomaly detection

Figure 4: The intuition behind PCA detection with simplified data. We plot only two dimensions from the Darkstar state variable feature. It is easy to see high correlation between these two dimensions. PCA determines the dominant normal pattern, separates it out, and makes it easier to identify anomalies.

Table 5: Low effective dimensionality of feature data. \( n = \) Dimensionality of feature vector \( y \); \( k = \) Dimensionality required to capture 95% of variance in the data. In all of our data, we have \( k \ll n \), exhibiting low effective dimensionality.
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

Data preprocessing

Training set: \( x^{(1)}, x^{(2)}, \ldots, x^{(m)} \)

Preprocessing (feature scaling/mean normalization):

\[
\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}
\]

Replace each \( x_j^{(i)} \) with \( x_j^{(i)} - \mu_j \).

If different features on different scales (e.g., \( x_1 \) = size of house, \( x_2 \) = number of bedrooms), scale features to have comparable range of values.

\[
x_j^{(i)} \quad \Rightarrow \quad \frac{x_j^{(i)} - \mu_j}{s_j}
\]
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

**Principal Component Analysis (PCA) algorithm**
Reduce data from $n$-dimensions to $k$-dimensions

Compute “covariance matrix”:

$$
\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)}) (x^{(i)})^T
$$

Compute “eigenvectors” of matrix $\Sigma$:

$$
[U, S, V] = \text{svd}(\Sigma);
$$

Andrew Ng
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

Principal Component Analysis (PCA) algorithm summary

- After mean normalization (ensure every feature has zero mean) and optionally feature scaling:
  \[ \text{Sigma} = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)})(x^{(i)})^T \]

- \([U,S,V] = \text{svd}(\text{Sigma});\]
- \(U_{\text{reduce}} = U(:,1:k);\]
- \(z = U_{\text{reduce}}' \ast x;\]
- \(x \in \mathbb{R}^n\)
Step 3: Machine learning

Principal Component Analysis (PCA)-based anomaly detection

\[ y_a = (I - PP^T)y \]

\[ P = [v_1, v_2, \ldots, v_k] \]

\[ \text{SPE} = \|y_a\|^2 > Q_\alpha \]
Step 3: Machine learning

Improving PCA detection results

• Applied Term Frequency / Inverse Document Frequency (TF-IDF)

\[ w_{i,j} \equiv y_{i,j} \log(n/df_j) \]

where \( df_j \) is total number of message groups that contain the j-th message type

• Using better similarity metrics and data normalization

\[ \mathcal{K}(x, y) = \frac{x \cdot y}{\sqrt{x \cdot x} \sqrt{y \cdot y}} \]
Step 4: Visualization

Figure 7: The decision tree visualization. Each node is the message type string (\# is the place holder for variables). The number on the edge is the threshold of message count, generated by the decision tree algorithm. Small boxes contain the labels from PCA, with a red 1 for abnormal and a green 0 for normal.
Methodology

void startTransaction() {
    ...
    LOG.info("starting" + transact);
}

Source Code

starting: xact 325 is PREPARING
prepare: xact 325 is COMMITTING
committed: xact 325 is COMMITTED

Raw Console Log

1. Log Parsing
   starting: xact (.*) is (.*)
   Message template

   type=1, tid=325, state=PREPARING
   type=2, tid=325, state=COMMITTING
   type=3, tid=325, state=COMMITTED

   Structured Log

2. Feature creation
   At time window 100
   COMMITTED
   PREPARING
   ABORTED
   COMMITTING
   State Ratio Vector

   325: 1 1 1 0 0 0 0 0 0
   326: 1 0 1 0 0 0 0 0
   327: 1 1 1 0 1 0 0 0
   Message Count Vectors

3. Anomaly detection

4. Visualization
   PCA Anomaly Detection
   Decision Tree
Evaluation

Dataset:

*Table 2.* Data sets used in evaluation. Nodes=Number of nodes in the experiments.

<table>
<thead>
<tr>
<th>System</th>
<th>Nodes</th>
<th>Messages</th>
<th>Log Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Darkstar</td>
<td>1</td>
<td>1,640,985</td>
<td>266 MB</td>
</tr>
<tr>
<td>Hadoop (HDFS)</td>
<td>203</td>
<td>24,396,061</td>
<td>2412 MB</td>
</tr>
</tbody>
</table>

- From Elastic Compute Cloud (EC2)
- 203 nodes of HDFS and 1 nodes of Darkstar
Evaluation

Parsing accuracy:

<table>
<thead>
<tr>
<th>System</th>
<th>Total Log</th>
<th>Failed</th>
<th>Failed %</th>
</tr>
</thead>
<tbody>
<tr>
<td>HDFS</td>
<td>24,396,061</td>
<td>29,636</td>
<td>0.121%</td>
</tr>
<tr>
<td>Darkstar</td>
<td>1,640,985</td>
<td>35</td>
<td>0.002%</td>
</tr>
</tbody>
</table>

Table 6: Parsing accuracy. Parse fails on a message when we cannot find a message template that matches the message and extract message variables.
Evaluation

Scalability:

Figure 5: Scalability of log parsing with number of nodes used. The x-axis is the number of nodes used, while the y-axis is the number of messages processed per minute. All nodes are Amazon EC2 high-CPU medium instances. We used the HDFS data set (described in (Table 3) with over 24 million lines. We parsed raw textual logs and generated the message count vector feature (see Section 4.2). Each experiment was repeated 4 times and the reported data point is the mean.

50 nodes, takes less than 3 minutes, less than 10 minutes with 10 nodes
Evaluation

Darkstar

• DarkMud Provided by the Darkstar team
• Emulate 60 user clients in the DarkMud virtual world performing random operations
• Run the experiment for 4800 seconds
• Injected a performance disturbance by capping the CPU available to Darkstar to 50% during time 1400 to 1800 sec
Evaluation

Darkstar - state ratio vectors

- 8 distinct values, including PREPARING, ACTIVE, COMMITTING, ABORTING and so on
- Ratio between number of ABORTING to COMMITTING increases from about 1:2000 to about 1:2
- Darkstar does not adjust transaction timeout accordingly
Evaluation

Darkstar - message count vectors

- 68,029 transaction ids reported in 18 different message types, $Y^m$ is $68,029 \times 18$
- PCA identifies the normal vectors: \{create, join txn, commit, prepareAndCommit\}
- Augmented each feature vector using the timestamp of the last message in that group
Evaluation

Hadoop

• Set up a Hadoop cluster on 203 EC2 nodes
• Run sample Hadoop map-reduce jobs for 48 hours
• Generate and processing over 200 TB of random datas
• Collect over 24 million lines of logs from HDFS
Evaluation

Hadoop - message count vectors

• Automatically chooses one identifier variable, the blockid, which is reported in 11,197,954 messages (about 50% of all messages) in 29 message types.

• $\mathbf{y}_m$ has a dimension of 575, $139 \times 29$
Evaluation

Hadoop - message count vectors

<table>
<thead>
<tr>
<th>#</th>
<th>Anomaly Description</th>
<th>Actual</th>
<th>Raw</th>
<th>TF-IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Namenode not updated after deleting block</td>
<td>4297</td>
<td>475</td>
<td>4297</td>
</tr>
<tr>
<td>2</td>
<td>Write exception client give up</td>
<td>3225</td>
<td>3225</td>
<td>3225</td>
</tr>
<tr>
<td>3</td>
<td>Write failed at beginning</td>
<td>2950</td>
<td>2950</td>
<td>2950</td>
</tr>
<tr>
<td>4</td>
<td>Replica immediately deleted</td>
<td>2809</td>
<td>2803</td>
<td>2788</td>
</tr>
<tr>
<td>5</td>
<td>Received block that does not belong to any file</td>
<td>1240</td>
<td>20</td>
<td>1228</td>
</tr>
<tr>
<td>6</td>
<td>Redundant addStoredBlock</td>
<td>953</td>
<td>33</td>
<td>953</td>
</tr>
<tr>
<td>7</td>
<td>Delete a block that no longer exists on data node</td>
<td>724</td>
<td>18</td>
<td>650</td>
</tr>
<tr>
<td>8</td>
<td>Empty packet for block</td>
<td>476</td>
<td>476</td>
<td>476</td>
</tr>
<tr>
<td>9</td>
<td>Receive block exception</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>10</td>
<td>Replication Monitor timedout</td>
<td>45</td>
<td>37</td>
<td>45</td>
</tr>
<tr>
<td>11</td>
<td>Other anomalies</td>
<td>108</td>
<td>91</td>
<td>107</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>16916</td>
<td>10217</td>
<td>16808</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>False Positive Description</th>
<th>Raw</th>
<th>TF-IDF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal background migration</td>
<td>1399</td>
<td>1397</td>
</tr>
<tr>
<td>2</td>
<td>Multiple replica (for task / job desc files)</td>
<td>372</td>
<td>349</td>
</tr>
<tr>
<td>3</td>
<td>Unknown Reason</td>
<td>26</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>1797</td>
<td>1746</td>
</tr>
</tbody>
</table>

Table 7: Detected anomalies and false positives using PCA on Hadoop message count vector feature. Actual is the number of anomalies labeled manually. Raw is PCA detection result on raw data, TF-IDF is detection result on data preprocessed with TF-IDF and normalized by vector length (Section 5).

- The first anomaly in Table 7 uncovered a bug that has been hidden in HDFS for a long time. No single error message indicating the problem.
- We do not have the problem that causes confusion in traditional grep based log analysis. `#:Got Exception while serving # to #:`
- Algorithm does report some false positives, which are inevitable. A few blocks are replicated 10 times instead of 3 times for the majority of blocks.
Online Detection

Two Stage Online Detection Systems
Stage 1: Frequent pattern based filtering

- **event trace**: a group of events that reports the same identifier.
- **session**: a subset of closely-related events in the same event trace that has a predictable duration.
- **duration**: the time difference between the earliest and latest timestamps of events in the session.
- **frequent pattern**: a session with its duration distribution:
  - 1) the session is frequent in many event traces;
  - 2) most (e.g., 99.95th percentile) of the session’s duration is less than $T_{\text{max}}$
- **$T_{\text{max}}$$^\ddagger$$^\ddagger$**: a user-specified maximum allowable detection latency
- **detection latency**: the time between an event occurring and the decision of whether the event is normal or abnormal
Stage 1: Frequent pattern based filtering

• **A. Combining time and sequence information**
  • *Step 1*: Use time gaps to find first session in each execution trace (coarsely)
    the time gap size is a configurable parameter
  • *Step 2*: Identify the dominant session
  • *Step 3*: Refine result using the frequent session and compute duration statistics

• **B. Estimating distributions of session durations**

power-law distribution
A. Stage 1 Pattern mining results

Table I
Frequent patterns mined. Pattern 3’s duration cannot be estimated because the durations are too small to capture in training set. Patterns 4–6 consist of only a single event each and thus have no durations.

<table>
<thead>
<tr>
<th>#</th>
<th>Frequent sessions</th>
<th>Duration in sec (%ile)</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>99.90</td>
<td>99.95</td>
</tr>
<tr>
<td>1</td>
<td>Allocated block, begin write</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>2</td>
<td>Done write, update block map</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>Delete block</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>Serving block</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>5</td>
<td>Read Exception (see text)</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>6</td>
<td>Verify block</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Online Detection - Evaluation

B. Detection precision and recall

<table>
<thead>
<tr>
<th>Table II</th>
<th>DETECTION PRECISION AND RECALL.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a) Varying ( \alpha ) while holding ( T_{max} = 60 )</td>
<td></td>
</tr>
<tr>
<td>( \alpha )</td>
<td>TP</td>
</tr>
<tr>
<td>----------</td>
<td>-----</td>
</tr>
<tr>
<td>0.0001</td>
<td>16,916</td>
</tr>
<tr>
<td><strong>0.001</strong></td>
<td><strong>16,916</strong></td>
</tr>
<tr>
<td>0.005</td>
<td>16,916</td>
</tr>
<tr>
<td>0.01</td>
<td>16,916</td>
</tr>
</tbody>
</table>

<p>| (b) Varying ( T_{max} ) while holding ( \alpha = 0.001 ) |</p>
<table>
<thead>
<tr>
<th>( T_{max} )</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>2,870</td>
<td>129</td>
<td>14,046</td>
<td>95.70%</td>
<td>16.97%</td>
</tr>
<tr>
<td>30</td>
<td>16,916</td>
<td>2,748</td>
<td>0</td>
<td>86.03%</td>
<td>100.00%</td>
</tr>
<tr>
<td><strong>60</strong></td>
<td><strong>16,916</strong></td>
<td><strong>2,748</strong></td>
<td><strong>0</strong></td>
<td><strong>86.03%</strong></td>
<td><strong>100.00%</strong></td>
</tr>
<tr>
<td>120</td>
<td>16,916</td>
<td>2,748</td>
<td>0</td>
<td>86.03%</td>
<td>100.00%</td>
</tr>
<tr>
<td>240</td>
<td>14,233</td>
<td>2,232</td>
<td>2,683</td>
<td>86.44%</td>
<td>84.14%</td>
</tr>
</tbody>
</table>
Online Detection - Evaluation

C. Detection latency

Figure 4. Detection latency and number of events kept in detector’s buffer
Online Detection - Evaluation

D. Comparison to offline results

<table>
<thead>
<tr>
<th>#</th>
<th>Anomaly Description</th>
<th>Actual</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Namenode not updated after deleting block</td>
<td>4297</td>
<td>4297</td>
<td>4297</td>
</tr>
<tr>
<td>2</td>
<td>Write exception client give up</td>
<td>3225</td>
<td>3225</td>
<td>3225</td>
</tr>
<tr>
<td>3</td>
<td>Write failed at beginning</td>
<td>2950</td>
<td>2950</td>
<td>2950</td>
</tr>
<tr>
<td>4</td>
<td>Replica immediately deleted</td>
<td>2809</td>
<td>2788</td>
<td>2809</td>
</tr>
<tr>
<td>5</td>
<td>Received block that does not belong to any file</td>
<td>1240</td>
<td>1228</td>
<td>1240</td>
</tr>
<tr>
<td>6</td>
<td>Redundant addStoredBlock</td>
<td>953</td>
<td>953</td>
<td>953</td>
</tr>
<tr>
<td>7</td>
<td>Delete a block that no longer exists on data node</td>
<td>724</td>
<td>650</td>
<td>724</td>
</tr>
<tr>
<td>8</td>
<td>Empty packet for block</td>
<td>476</td>
<td>476</td>
<td>476</td>
</tr>
<tr>
<td>9</td>
<td>Receive block exception</td>
<td>89</td>
<td>89</td>
<td>89</td>
</tr>
<tr>
<td>10</td>
<td>Replication monitor timeout</td>
<td>45</td>
<td>45</td>
<td>45</td>
</tr>
<tr>
<td>11</td>
<td>Other anomalies</td>
<td>108</td>
<td>107</td>
<td>108</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>16916</strong></td>
<td><strong>16808</strong></td>
<td><strong>16916</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>False Positive Description</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Normal background migration</td>
<td>1397</td>
<td>1403</td>
</tr>
<tr>
<td>2</td>
<td>Multiple replica (for task / job desc files)</td>
<td>349</td>
<td>368</td>
</tr>
<tr>
<td></td>
<td><strong>Total</strong></td>
<td><strong>1746</strong></td>
<td><strong>1771</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>Ambiguous Case</th>
<th>Offline</th>
<th>Online</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(see Section VII-D)</td>
<td>0</td>
<td>977</td>
</tr>
</tbody>
</table>
Conclusion

- Propose a general approach to problem detection via the analysis of console logs.
- Use source code as a reference to understand the structure of console logs to parse logs accurately.
- Use parsed logs to construct powerful features capturing both global states and individual operation sequences.
- Use simple algorithms such as PCA yield promising anomaly detection results.
- Adopt a two-stage approach which uses frequent pattern to filter out normal events while using PCA detection to detect the anomalies in an online setting.
Thanks for your attention
Q&A