Automated Runtime Data Analysis for System Reliability Management

HE, Pinjia
Supervisor: Prof. Michael R. Lyu

2018/02/08
Modern systems are serving many aspects of our daily life
Popular modern systems

Search engine
Cloud services

Online chatting
Office software
And many others...
System reliability is very important
Real-World Revenue Loss

148,213 websites
121,176 unique domains

Real-World Revenue Loss

Lloyd's Estimates the Impact of a U.S. Cloud Outage at $19 Billion

By: Sean Michael Kerner | January 24, 2018

A joint research report from insurance provider Lloyd's of London and the American Institutes for Research (AIR), looks at the potential costs related to a major public cloud outage in the U.S.

As organizations around the world increasingly rely on the cloud, the impact of a public cloud failure is something that insurance companies are now concerned about. A 67-page report released on Jan. 23 from Lloyd's of London and AIR Worldwide provides some insight and estimates on the potential losses from a major cloud services outage—and the numbers are large.

According to the report, a cyber-incident that impacted the operations of one of the top three public cloud providers in the U.S. for three to six days, could result in total losses of up to $19 billion. Of those loses, only $1.1 to $3.5 billion would be insured, leaving organizations left to cover the rest of the costs.

Reliability management of modern systems is important, but challenging.
Modern systems are becoming large-scale in size

[Image from: http://www.lancaster.ac.uk/scc/research/distributed-systems/]
Modern systems are complex in structure

[Image from: http://www.smashingbuzz.com/2015/01/ultimate-github-features]
Traditional engineering techniques are often not sufficient. Automated runtime data analysis is in need.
Automated runtime data analysis for system reliability management
Automated runtime data analysis for system reliability management

- User Info
- System Logs
- QoS Values

Operational Issues Prioritization
QoS Prediction

Reliability Management
Automated runtime data analysis for system reliability management
Thesis contributions

Evaluation study on log parsing [DSN’16] (Chapter 3)
Parallel log parsing [TDSC’17] (Chapter 4)
Online log parsing [ICWS’17] (Chapter 5)
Thesis contributions

Evaluation study on log parsing [DSN’16] (Chapter 3)
- Reviews and evaluate four representative log parsers
  - A case study of the effectiveness of log parsers on log mining
  - Six findings and open-source toolkit

Parallel log parsing [TDSC’17] (Chapter 4)
- The first parallel log parsing framework
- Specially-designed heuristic rules and clustering algorithm
- Evaluate on real-world data and large-scale synthetic data

Online log parsing [ICWS’17] (Chapter 5)
- Online parser Drain via fixed depth tree
- 51.85% ~ 81.47% efficiency improvement with comparable accuracy
- Open-source
Thesis contributions

Operational issues prioritization (Chapter 6)
- An operational issue prioritization framework POI
- Coarse-grained clustering and fine-grained clustering
- Novel weighting method Inverse Cardinality (IC)

QoS prediction [ICWS’14] (Chapter 7)
- Hierarchical matrix factorization model
- Location of both users and services
Outline

• **Topic 1:** Evaluation study on log parsing

• **Topic 2:** Parallel log parsing for large-scale log data

• **Topic 3:** Online log parsing via fixed depth tree

• Conclusion and future work
Outline

• **Topic 1: Evaluation study** on log parsing

• **Topic 2: Parallel log parsing** for large-scale log data

• **Topic 3: Online log parsing** via fixed depth tree

• Conclusion and future work
Logs are widely-employed to enhance the system reliability by log analysis.
Log Analysis

Detecting largescale system problems by mining console logs [SOSP’09]

Log Clustering based Problem Identification for Online Service Systems [ICSE’16]

Leveraging existing instrumentation to automatically infer invariant-constrained models [FSE’11]

Assisting developers of big data analytics applications when deploying on hadoop clouds [ICSE’13]

Structured comparative analysis of systems logs to diagnose performance problems [NSDI’12]

Enhancing failure diagnosis with proactive logging [OSDI’12]
Log Analysis contains two steps:
Log Parsing and Log Mining
Log Analysis: log parsing & log mining

### Raw Log Messages

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2008-11-11 03:40:58</td>
<td>BLOCK* NameSystem.allocateBlock: /user/root/randtxt4/_temporary/_task_200811101024_0010_m_000011_0/part-00011.blk_904791815409399662</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>PacketResponder 0 for block blk_904791815409399662 terminating</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>Received block blk_904791815409399662 of size 67108864 from /10.250.18.114</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>PacketResponder 1 for block blk_904791815409399662 terminating</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>Received block blk_904791815409399662 of size 67108864 from /10.251.43.210</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.251.43.210:50010 is added to blk_904791815409399662 size 67108864</td>
</tr>
<tr>
<td>2008-11-11 03:41:48</td>
<td>BLOCK* NameSystem.addStoredBlock: blockMap updated: 10.250.18.114:50010 is added to blk_904791815409399662 size 67108864</td>
</tr>
<tr>
<td>2008-11-11 08:30:54</td>
<td>Verification succeeded for blk_904791815409399662</td>
</tr>
</tbody>
</table>
The goal of log parsing is to distinguish between constant part and variable part from the log contents.
Log Parsing: a clustering problem

Event 1
Event 2
Event 3
Event 4

Event 1
Event 2
Event 3
Event 4
Manual maintenance of log event is difficult, even with the help of regular expression

- **The volume of log is growing rapidly.** (e.g., 50 GB/h [Mi TPDS’13])

- **Developer may not understand the logging purpose.** (open source components [Xu SOSP’09])

- **Log statements in modern systems update frequently.** (e.g., hundreds of new statements a month [Xu PhD Thesis’10])
Automated log parsing is highly in demand
State-of-the-art Log Parsing Methods

- **SLCT**: Simple Logfile Clustering Tool [IPOM’03]
- **IPLoM**: Iterative Partitioning Log Mining [KDD’09, TKDE’12]
- **LKE**: Log Key Extraction [ICDM’09]
- **LogSig**: Log Signature Extraction [CIKM’11]
Will the performance of log parsers affect the anomaly detection results?
• Case study on real-world anomaly detection task [SOSP’09]

• 11,175,629 HDFS logs
• 575,061 HDFS blocks
• 16,838 anomalies
Accuracy Metric

• Parsing accuracy: **F-measure** of clustering algorithm

• **F-measure** = \(2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}\)

• Precision = \(\frac{\text{TP}}{\text{TP} + \text{FP}}\)  

• Recall = \(\frac{\text{TP}}{\text{TP} + \text{FN}}\)

• **TP**: assigns two logs with the same log event to the same cluster

• **TN**: assigns two logs with different log events to different clusters

• **FP**: assigns two logs with different log events to the same cluster
- **Parsing Accuracy**: F-measure
- **Report Anomaly**: #anomalies reported
- **Detected Anomaly**: #true anomalies detected
- **False Alarm**: #wrongly detected anomalies

<table>
<thead>
<tr>
<th></th>
<th>Parsing Accuracy</th>
<th>Reported Anomaly</th>
<th>Detected Anomaly</th>
<th>False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCT</td>
<td>0.83</td>
<td>18,450</td>
<td>10,935 (64%)</td>
<td>7,515 (40%)</td>
</tr>
<tr>
<td>LogSig</td>
<td>0.87</td>
<td>11,091</td>
<td>10,678 (63%)</td>
<td>413 (3.7%)</td>
</tr>
<tr>
<td>IPLoM</td>
<td>0.99</td>
<td>10,998</td>
<td>10,720 (63%)</td>
<td>278 (2.5%)</td>
</tr>
<tr>
<td>Ground truth</td>
<td>1.00</td>
<td>11,473</td>
<td>11,195 (66%)</td>
<td>278 (2.4%)</td>
</tr>
</tbody>
</table>
Finding: Log parsing is important because log mining is effective only when the parsing accuracy is high enough.
<table>
<thead>
<tr>
<th></th>
<th>Parsing Accuracy</th>
<th>Reported Anomaly</th>
<th>Detected Anomaly</th>
<th>False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCT</td>
<td>0.83</td>
<td>18,450</td>
<td>10,935 (64%)</td>
<td>7,515 (40%)</td>
</tr>
<tr>
<td>LogSig</td>
<td>0.87</td>
<td>11,091</td>
<td>10,678 (63%)</td>
<td>413 (3.7%)</td>
</tr>
<tr>
<td>IPLoM</td>
<td>0.99</td>
<td>10,998</td>
<td>10,720 (63%)</td>
<td>278 (2.5%)</td>
</tr>
<tr>
<td>Ground truth</td>
<td>1.00</td>
<td>11,473</td>
<td>11,195 (66%)</td>
<td>278 (2.4%)</td>
</tr>
<tr>
<td></td>
<td>Parsing Accuracy</td>
<td>Reported Anomaly</td>
<td>Detected Anomaly</td>
<td>False Alarm</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>SLCT</td>
<td>0.83</td>
<td>18,450</td>
<td>10,935 (64%)</td>
<td>7,515 (40%)</td>
</tr>
<tr>
<td>LogSig</td>
<td>0.87</td>
<td>11,091</td>
<td>10,678 (63%)</td>
<td>413 (3.7%)</td>
</tr>
</tbody>
</table>

### Diagram Description

**Original SLCT**
- Deleting block file /mnt/hadoop/dfs/data/current/subdir28/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir19/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir52/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir16/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir60/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir54/
- Deleting block file /mnt/hadoop/dfs/data/current/subdir32/

**Refined SLCT**
- Deleting block file

**Reported Anomaly Block:**

```
65 -> 575,061
1 2 2 2   ...
1 2 2 2   ...
1 0 0 3   ...
```

**Detected Anomaly Block:**

```
24 -> 575,061
1 1   ....
1 2   ....
1 0   ....
```
## Finding

Log mining is sensitive to some critical events. Errors in parsing 1 log event could even cause nearly an order of magnitude performance degradation in log mining.

### Table

<table>
<thead>
<tr>
<th></th>
<th>Parsing Accuracy</th>
<th>Reported Anomaly</th>
<th>Detected Anomaly</th>
<th>False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCT</td>
<td>0.83</td>
<td>18,450</td>
<td>10,935 (64%)</td>
<td>7,515 (40%)</td>
</tr>
<tr>
<td>LogSig</td>
<td>0.87</td>
<td>11,091</td>
<td>10,678 (63%)</td>
<td>413 (3.7%)</td>
</tr>
</tbody>
</table>

### Diagram

- Deleting block file `/mnt/hadoop/dfs/data/current/subdir28/`
- Deleting block file `/mnt/hadoop/dfs/data/current/subdir19/`
- Starting thread to transfer block to
- Receiving block src: dest:

### Parsing Accuracy

<table>
<thead>
<tr>
<th></th>
<th>Parsing Accuracy</th>
<th>Reported Anomaly</th>
<th>Detected Anomaly</th>
<th>False Alarm</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogSig</td>
<td>0.91</td>
<td>11,539</td>
<td>10,746 (64%)</td>
<td>793 (6.8%)</td>
</tr>
</tbody>
</table>
Outline

• **Topic 1:** Evaluation study on log parsing

• **Topic 2:** Parallel log parsing for large-scale log data

• **Topic 3:** Online log parsing via fixed depth tree

• Conclusion and future work
Why we need parallel log parsers?
Motivations & Contributions

**Weakness** of existing parsers

- Existing log parsers do **not consistently** obtain **high accuracy** on all datasets.

- When logs grow to a **large scale**, existing parsers **fail to complete** in reasonable time.
Motivations & Contributions

Weakness of existing parsers

• Existing log parsers do not consistently obtain high accuracy on all datasets.

POP achieves the highest parsing accuracy on all datasets.

• When logs grow to a large scale, existing parsers fail to complete in reasonable time.

POP can handle 200m HDFS logs in 7 mins, while the state-of-the-art needs more than half an hour.
Framework of POP

1. Distributed File System
2. Spark Cluster
3. Worker
4. Worker
5. Worker
6. Preprocess by Domain Knowledge (Step 1)
7. Partition by Log Message Length (Step 2)
8. Recursively Partition by Token Position (Step 3)
9. Generate Log Events (Step 4)
10. Merge Groups by Log Event (Step 5)
11. Spark Cluster
12. Spark Driver
13. Distributed File System
14. Spark Cluster
15. Spark Cluster
Novelty of POP

Accuracy

Efficiency

Distributed File System

Spark Cluster

Worker

Preprocess by Domain Knowledge
Step 1

Partition by Log Message Length
Step 2

Recursively Partition by Token Position
Step 3

Generate Log Events
Step 4

Merge Groups by Log Event
Step 5

Spark Driver

Accuracy

Efficiency
Step 1: Preprocess by domain knowledge

- **Prune** variable parts according to simple regular expressions

```
Received block  blk_904791815409399662  of size 67108864 from /10.251.43.210
```

```
Received block of size 67108864 from /10.251.43.210
```
Step 2: Partition by log message length

- Partition logs into different groups based on log message length

<table>
<thead>
<tr>
<th>Log message length</th>
<th>1088 bytes sent, 0128 bytes received, lifetime 00:03</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1024 bytes sent, 2056 bytes received, lifetime 00:10</td>
</tr>
<tr>
<td>Send file file_01</td>
<td></td>
</tr>
<tr>
<td>Send file file_02</td>
<td></td>
</tr>
<tr>
<td>Receive file file_03</td>
<td></td>
</tr>
<tr>
<td>Connect to IP_01</td>
<td></td>
</tr>
</tbody>
</table>

Send file file_01
Send file file_02
Receive file file_03
Connect to IP_01
Step 3: Recursively partition by token position

- Find **split token position**
- **Recursively** partition a log group
- Stop until all log groups are **complete groups**
Step 3: Recursively partition by token position

- Find **split token position**

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Send</td>
<td>file</td>
<td>file_01</td>
</tr>
<tr>
<td></td>
<td>Send</td>
<td>file</td>
<td>file_02</td>
</tr>
<tr>
<td></td>
<td>Receive</td>
<td>file</td>
<td>file_03</td>
</tr>
<tr>
<td></td>
<td>Connect</td>
<td>to</td>
<td>IP_01</td>
</tr>
</tbody>
</table>

1: {Send, Receive, Connect}
2: {file, to}
3: {file_01, file_02, file_03, IP_01}

1: 3 distinct tokens
2: 2 distinct tokens
3: 4 distinct tokens
Step 3: Recursively partition by token position

- **Find** split token position
- **Recursively** partition a log group
Step 3: Recursively partition by token position

• **Find** split token position
• **Recursively** partition a log group
• Stop until all log groups are complete groups
Step 3: Recursively partition by token position

• Stop until all log groups are complete groups

Group goodness = \( \frac{\text{#token position with one distinct token}}{\text{log message length}} \)

- \( \frac{1}{3} \approx 0.33 \)
- \( \frac{2}{3} \approx 0.66 \)

• Compare with a group threshold \( g_s \) (e.g., 0.5)
Step 4: Generate log events

- Inspect the tokens in each **token position** of each log, calculate the number of **distinct** tokens.

- Send file `file_01`
- Send file `file_02`
- Receive file `file_03`
- Receive file `file_04`

- 1088 bytes sent, 0128 bytes received, lifetime 00:03
- 1024 bytes sent, 2056 bytes received, lifetime 00:10

* bytes sent, * bytes received, lifetime *
Step 5: Merge groups by log event

- **Hierarchical clustering on log events**, instead of log messages

1. Send configuration file
2. Send network file

1. Send user interface file
2. Send set up file

```
Send * file
```

```
Send ** file
```

```
Send * file
1. Send configuration file
2. Send network file
3. Send user interface file
4. Send set up file
```
Parallelization

- Log message lengths in Step 2

1088 bytes sent, 0128 bytes received, lifetime 00:03
Send file file_01

1024 bytes sent, 2056 bytes received, lifetime 00:10
Send file file_02

Connect to IP_01
Stop process
Receive file file_03
**Data sets**

- Data set (supercomputer, distributed system, standalone software)

<table>
<thead>
<tr>
<th>System</th>
<th>Description</th>
<th>#Logs</th>
<th>Length</th>
<th>#Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>BGL</td>
<td>BlueGene/L Supercomputer</td>
<td>4,747,963</td>
<td>10~102</td>
<td>376</td>
</tr>
<tr>
<td>HPC</td>
<td>High Performance Cluster (Los Alamos)</td>
<td>433,490</td>
<td>6~104</td>
<td>105</td>
</tr>
<tr>
<td>Proxifier</td>
<td>Proxy Client</td>
<td>10,108</td>
<td>10~27</td>
<td>8</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop File System</td>
<td>11,175,629</td>
<td>8~29</td>
<td>29</td>
</tr>
<tr>
<td>Zookeeper</td>
<td>Distributed System Coordinator</td>
<td>74,380</td>
<td>8~27</td>
<td>80</td>
</tr>
</tbody>
</table>

• Randomly select 2,000 logs from each data set
RQ1: Accuracy    RQ2: Efficiency

• **Accuracy results:**

<table>
<thead>
<tr>
<th></th>
<th>BGL</th>
<th>HPC</th>
<th>HDFS</th>
<th>Zookeeper</th>
<th>Proxifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLCT</td>
<td>0.94</td>
<td>0.86</td>
<td>0.93</td>
<td>0.92</td>
<td>0.89</td>
</tr>
<tr>
<td>IPlom</td>
<td>0.99</td>
<td>0.64</td>
<td>1.00</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>LKE</td>
<td>0.70</td>
<td>0.17</td>
<td>0.96</td>
<td>0.82</td>
<td>0.81</td>
</tr>
<tr>
<td>LogSig</td>
<td>0.98</td>
<td>0.87</td>
<td>0.93</td>
<td>0.99</td>
<td>0.84</td>
</tr>
<tr>
<td>POP</td>
<td>0.99</td>
<td>0.95</td>
<td>1.00</td>
<td>0.99</td>
<td>1.00</td>
</tr>
</tbody>
</table>
• Evaluate the running time of log parsing methods on all data sets by **varying the number of raw logs.**

<table>
<thead>
<tr>
<th></th>
<th>BGL</th>
<th>HPC</th>
<th>HDFS</th>
<th>Zookeeper</th>
<th>Proxifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size</td>
<td>400k</td>
<td>600k</td>
<td>1k</td>
<td>4k</td>
<td>600k</td>
</tr>
<tr>
<td>Time</td>
<td>4k</td>
<td>3k</td>
<td>10k</td>
<td>8k</td>
<td>1200</td>
</tr>
<tr>
<td>Time</td>
<td>40k</td>
<td>15k</td>
<td>100k</td>
<td>16k</td>
<td>2400</td>
</tr>
<tr>
<td>Time</td>
<td>400k</td>
<td>75k</td>
<td>1m</td>
<td>32k</td>
<td>4800</td>
</tr>
<tr>
<td>Time</td>
<td>4m</td>
<td>375k</td>
<td>10m</td>
<td>64k</td>
<td>9600</td>
</tr>
</tbody>
</table>
Efficiency experiments (real-world datasets):

- Running time (y-axis)
- Slope
RQ1: Accuracy    RQ2: Efficiency

- SLCT
- LKE
- IPLoM
- LogSig
- SinglePOP
- POP

Time (Sec)

Log Size of Sample Datasets from BGL

Log Size of Sample Datasets from HPC

Log Size of Sample Datasets from Zookeeper

Log Size of Sample Datasets from Proxifier
RQ1: Accuracy  RQ2: Efficiency

- Efficiency experiments (synthetic datasets):

![Graphs showing Time (Sec) vs Log Size (m) for synthetic datasets from HDFS and BGL with lines for SLCT, IPlOM, and POP]
Outline

• **Topic 1:** Evaluation study on log parsing

• **Topic 2:** Parallel log parsing for large-scale log data

• **Topic 3:** Online log parsing via fixed depth tree

• Conclusion and future work
Why we need online log parsers?
Motivations

• **Offline log parsers**
  – Log event changes

• **Modern system structure**
  – Log collection works in a streaming manner
An **online log parser** is in demand
Framework of Online Parser

2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114

blk_90 -> Received block * of size * from *
Framework of Drain

2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114

Fixed depth tree

blk_90 -> Received block * of size * from *
Novelty of Drain

2008-11-11 03:41:48 Received block blk_90 of size 67108864 from /10.250.18.114

- Encodes heuristic rules in the tree to conduct pre-clustering.
- The depth can be fixed by a pre-defined parameter depth.

blk_90 -> Received block * of size * from *
Depth

Under-parsed

Over-parsed

Send file file_01
Send file file_01
Receive file file_02

Send file run.py
Send file boot.py
Framework of Drain

A List of Log Groups

Length: 4
Send

Length: 5
Receive

...  
Length: 10
Starting

Log Group
Log Event: Receive from node *
Log IDs: [1, 23, 25, 46, 345, ...]

Fixed depth tree (depth=3)
Update of Drain

Fixed depth tree (depth=4)

Send block

Log Event: Send block 44
Log IDs: [1]

Receive file 01
Length is 3
First token is Receive
NOT match!
Need to update the tree.
Update of Drain

Fixed depth tree (depth=4)
### Accuracy results:

<table>
<thead>
<tr>
<th></th>
<th>BGL</th>
<th>HPC</th>
<th>HDFS</th>
<th>Zookeeper</th>
<th>Proxifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LKE</strong></td>
<td>0.67</td>
<td>0.17</td>
<td>0.57</td>
<td>0.78</td>
<td>0.85</td>
</tr>
<tr>
<td><strong>IPLoM</strong></td>
<td>0.99</td>
<td>0.65</td>
<td>0.99</td>
<td>0.99</td>
<td>0.85</td>
</tr>
<tr>
<td><strong>Online Log Parsers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SHISO</strong></td>
<td>0.87</td>
<td>0.53</td>
<td>0.93</td>
<td>0.68</td>
<td>0.85</td>
</tr>
<tr>
<td><strong>Spell</strong></td>
<td>0.98</td>
<td>0.82</td>
<td>0.87</td>
<td>0.99</td>
<td>0.87</td>
</tr>
<tr>
<td><strong>Drain</strong></td>
<td><strong>0.99</strong></td>
<td><strong>0.84</strong></td>
<td><strong>0.99</strong></td>
<td><strong>0.99</strong></td>
<td><strong>0.84</strong></td>
</tr>
<tr>
<td><strong>Offline Log Parsers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Efficiency experiments:**

<table>
<thead>
<tr>
<th></th>
<th>BGL</th>
<th>HPC</th>
<th>HDFS</th>
<th>Zookeeper</th>
<th>Proxifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Offline Log Parsers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LKE</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
<td>8888.49</td>
</tr>
<tr>
<td>IPLoM</td>
<td>140.57</td>
<td>12.74</td>
<td>333.03</td>
<td>2.17</td>
<td>0.38</td>
</tr>
<tr>
<td><strong>Online Log Parsers</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SHISO</td>
<td>10964.55</td>
<td>582.14</td>
<td>6649.23</td>
<td>87.61</td>
<td>8.41</td>
</tr>
<tr>
<td>Spell</td>
<td>447.14</td>
<td>47.28</td>
<td>676.45</td>
<td>5.27</td>
<td>0.87</td>
</tr>
<tr>
<td>Drain</td>
<td>115.96</td>
<td>8.76</td>
<td>325.7</td>
<td>1.81</td>
<td>0.27</td>
</tr>
<tr>
<td>Improvement</td>
<td><strong>74.07%</strong></td>
<td><strong>81.47%</strong></td>
<td><strong>51.85%</strong></td>
<td><strong>65.65%</strong></td>
<td><strong>68.97%</strong></td>
</tr>
</tbody>
</table>
Parsers

If you are not familiar with log parser, please check the Principles of Parsers. The codes are here.

- SLCT (Simple Logfile Clustering Tool): A Data Clustering Algorithm for Mining Patterns from Event Logs (SLCT is wrapped around the C source code provided by the author.)
- IPLoM (Iterative Partitioning Log Mining): A Lightweight Algorithm for Message Type Extraction in System Application Logs
- LKE (Log Key Extraction): Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis
- LogSig: LogSig: Generating System Events from Raw Textual Logs
- Drain: Drain: An Online Log Parsing Approach with Fixed Depth Tree
- POP: a parallel log parsing method optimized on top of Spark.

Data

In data, there are 5 datasets for you to play with. Each dataset contains several text files.

- rawlog.log: The raw log messages with ID. "ID\tword1 word2 word3"
- template[0-9]+: The log messages belong to a certain template.
- templates: The text of templates.

Quick Start

Input: A raw log file. Each line of the file follows "ID\tword1 word2 word3"
Output: Two parts. One is split log messages (only contains log ID) in different text files. The other is the templates file which contains all templates.

Examples: Before running the examples, please copy the parser source file to the same directory.

- Example1: This file is a simple example to demonstrate the usage of LogSig. The usage of other log parsers is similar.
- Example2: This file is to demonstrate the usage of POP.
- Example3: This file is used to evaluate the performance of LogSig. It iterates 10 times and record several important

Parsers are open source on github.com/logpai/logparser
Outline

- **Topic 1**: Evaluation study on log parsing

- **Topic 2**: Parallel log parsing for large-scale log data

- **Topic 3**: Online log parsing via fixed depth tree

- Conclusion and future work
Conclusion

Contributions

– Evaluation study of log parsing
  • Six insightful findings and an open-source log parsing toolkit

– Parallel log parsing for large-scale log data
  • A parallel log parsing framework POP

– Online log parsing
  • An online log parsing method Drain based on fixed depth tree
Conclusion

Contributions

– Evaluation study of log parsing
  • Six insightful findings and an open-source log parsing toolkit

– Parallel log parsing
  • A parallel log parsing framework POP built on top of Spark

– Online log parsing
  • An online log parsing method Drain based on fixed depth tree

– Operational issues prioritization
  • An operational issues prioritization method POI via hierarchical clustering

– Location-based QoS prediction
Future work

Parameter-free Online log parser

– An online log parser that automatically tunes the parameters
Publications (1)

Journal


Conference


Publications (2)


Thank you!

Q&A