LogParse: Making Log Parsing Adaptive through Word Classification

Weibin Meng, Ying Liu, Federico Zaiter, Shenglin Zhang, Yihao Chen, Yuzhe Zhang, Yichen Zhu, En Wang, Ruizhi Zhang, Shimin Tao, Dian Yang, Rong Zhou, Dan Pei
Background

■ Internet provide various types of services
■ The traffic is growing rapidly.

Traffic will increase more than three times

Source: Cisco VNI Global IP

<table>
<thead>
<tr>
<th>Year</th>
<th>Traffic (PB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>122</td>
</tr>
<tr>
<td>2018</td>
<td>156</td>
</tr>
<tr>
<td>2019</td>
<td>201</td>
</tr>
<tr>
<td>2020</td>
<td>254</td>
</tr>
<tr>
<td>2021</td>
<td>319</td>
</tr>
<tr>
<td>2022</td>
<td>396</td>
</tr>
</tbody>
</table>
Stability of services are becoming more and more important.

Monitor services to keep stability
Service logs

- Logs are the most valuable data for service management
- Logs record a vast range of events (7*24) of services
- Every service generates logs

<table>
<thead>
<tr>
<th>Types</th>
<th>Timestamps</th>
<th>Detailed messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch</td>
<td>Jul 10 19:03:03</td>
<td>Interface te-1/1/59, changed state to down</td>
</tr>
<tr>
<td>Supercomputer</td>
<td>Jun 4 6:45:50</td>
<td>RAS KERNEL INFO 87 L3 EDRAM dcr 0x0157 detected and corrected over 27362 seconds</td>
</tr>
<tr>
<td>Router</td>
<td>Jul 11 11:05:07</td>
<td>Neighbour(rid:10.231.0.43, addr:10.231.39.61) on vlan23, changed state from Exchange to Loading</td>
</tr>
</tbody>
</table>
Log parsing

- Log analysis → Log-based service management
- Log analysis contains two steps\(^1\):
  - Log Parsing and Log Mining
- Log parsing effects the performance of log analysis

---

Log parsing

- An unstructured log is “printf”ed by services.
- The goal of log parsing is to distinguish between constant part and variable part.

Raw logs:
- Interface ae3, changed state to down
- Vlan-interface vl22, changed state to down

Templates:
- Interface *, changed state to down
- Vlan-interface *, changed state to down

Templates consist of constant parts.
Adaptiveness is important for log parsing

- Goal: match any types of logs

- Intra-service adaptiveness 

- Cross-service adaptiveness 

Traditional log parsing approaches or don’t support intra-service adaptiveness, or do not support cross-service adaptiveness, or both.
Intra-service Adaptiveness

- **Intra-service adaptiveness**
  - Software/firmware upgrades can generate new types of logs
  - New logs cannot match any existing templates

**Historical logs:**
- \( L_1 \): Interface \( ae3 \), changed state to down
- \( L_2 \): Vlan-interface \( vl22 \), changed state to down
- \( L_3 \): Interface \( ae3 \), changed state to up
- \( L_4 \): Interface \( ae1 \), changed state to down

**Real-time logs:**
- \( L_5 \): Interface \( ae1 \), changed state to up
- \( L_6 \): Vlan-interface \( vl22 \), changed state to up

**Template extraction:**
- \( T_1 \): Interface *, changed state to down
- \( T_2 \): Vlan-interface *, changed state to down
- \( T_3 \): Interface *, changed state to up

**Template update:**
- \( T_4 \): ??

**Template match:**
- \( L_1 \rightarrow T_1, ae3 \)
- \( L_2 \rightarrow T_2, vl22 \)
- \( L_3 \rightarrow T_3, ae3 \)
- \( L_4 \rightarrow T_1, ae1 \)
- \( L_5 \rightarrow T_3, ae1 \)
- \( L_6 \rightarrow ?? \)
Intra-service Adaptiveness

- Traditional log parsing methods:
  - Drain (ICWS’17), FT-tree (IWQoS’18) which claimed to support template update
  - LogSig (CIKM’11), Spell (ICDM’16), IPLoM (KDD’09) don’t support template update

**Historical logs:**
- L₁. Interface ae₃, changed state to down
- L₂. Vlan-interface vl₂₂, changed state to down
- L₃. Interface ae₃, changed state to up
- L₄. Interface ae₁, changed state to down

**Real-time logs:**
- L₅. Interface ae₁, changed state to up
- L₆. Vlan-interface vl₂₂, changed state to up

**Template extraction:**
- T₁. Interface *, changed state to down
- T₂. Vlan-interface *, changed state to down
- T₃. Interface *, changed state to up
- T₄. Interface *, changed state to up

**Template update:**
- T₄. ??

**Template match:**
- L₁→T₁, ae₃
- L₂→T₂, vl₂₂
- L₃→T₃, ae₃
- L₄→T₁, ae₁
- L₅→T₃, ae₁
- L₆→? ??

When face new types of logs, all traditional methods achieve low accuracies

---

Traditional template extraction results when all the logs are used for training:

- Drain: ??
- Spell: ??
- IPLoM: ??
- FT-tree: ??
- LogSig: ??

Traditional template extraction results when only 10% of logs are used for training:

- Drain: ??
- Spell: ??
- IPLoM: ??
- FT-tree: ??
- LogSig: ??
Cross-service Adaptiveness

Observation:
- No enough historical logs when a brand new service goes online

Aim:
- **A model** trained by service A is also suitable for service B

- **Cross-service adaptive is for models rather than template sets.**
- Templates are generated by trained model.
Log compression

- Log parsing without adaptiveness limit many log analysis applications
- Requires a corresponding template for any given log
- Log Compression (An application)
  - Short-term storage (real-time queries)
    - Template index + Variables
  - Long-term storage
  - LogParse + Traditional method (e.g., zip)

Historical logs:
- L₁: Interface ae3, changed state to down
- L₂: Vlan-interface vl22, changed state to down
- L₃: Interface ae3, changed state to up
- L₄: Interface ae1, changed state to down

Real-time logs:
- L₅: Interface ae1, changed state to up
- L₆: Vlan-interface vl22, changed state to up

Short term storage by LogParse:
- L₁→T₁ ,ae3
- L₂→T₂ ,vl22
- L₃→T₃ ,ae3
- L₄→T₄ ,ae1
- L₅→T₃ ,ae1
- L₆→T₄ ,vl22

Long-term compression
Idea

Observation:

Operators usually distinguish variables based on features of words.

Historical logs:
- L₁. Interface ae3, changed state to down
- L₂. Vlan-interface vl22, changed state to down
- L₃. Interface ae3, changed state to up
- L₄. Interface ae1, changed state to down

Real-time logs:
- L₅. Interface ae1, changed state to up
- L₆. Vlan-interface vl22, changed state to up

Mixed characters and numbers are usually variables

Letters are usually template words

A log parsing problem $\rightarrow$ A word classification problem
LogParse Workflow

**Offline learning:**
- Prepare training word sets and train word classifier

**Online log parsing:**
- Match logs and update template sets

Toolkit: [https://github.com/WeibinMeng/LogParse](https://github.com/WeibinMeng/LogParse)

An adaptive framework for online log parsing
**Offline Learning**

1. Extract templates
2. Prepare training sets
3. Represent words
4. Train classifier

- Historical Logs
- Real-time logs
- Template extraction
- Word labels
- Word representation
- Word classifier
- Template word combination
- Word representation
- Word Classification

Offline learning

Online matching

2021/1/27

Weibin Meng
Template extraction

- Extract templates by traditional log parsing methods
  - Generate accurate templates (in offline stage)
- Unsupervised methods
- Use the results as the initial template set

**Rawlogs:**
- L₁: Interface ae3, changed state to down
- L₂: Vlan-interface vlan22, changed state to down
- L₃: Interface ae3, changed state to up.
- L₄: Interface ae1, changed state to down

**Templates:**
- T₁: Interface *, changed state to down
- T₂: Vlan-interface *, changed state to down
- T₃: Vlan-interface *, changed state to up
Prepare training sets

- Distinguish variable/template words
  - Variable words: words in logs but not in templates
  - Template words: words in templates

**Rawlogs:**
- L₁. Interface ae3, changed state to down
- L₂. Vlan-interface vlan22, changed state to down
- L₃. Interface ae3, changed state to up.
- L₄. Interface ae1, changed state to down

**Templates:**
- T₁. Interface *, changed state to down
- T₂. Vlan-interface *, changed state to down
- T₃. Vlan-interface *changed state to up
Word representation

- Machine learning algorithms require structured data
- Present each word by using a character-level count vector
  - The set of characters is fixed → fixed dimensionality
  - e.g., 128 characters in ASCII

```
Templates words: Interface changed state to Vlan-interface down up

Variables words: ae3, ae1, vlan22
```

Character count vector: `a - z, A - Z, 0 - 10, symbol label`

- `[1, 2, 4, x, 0, 0, ..., x, x, x]` template
- `[x, x, x, x, 1, 2, ..., x, x, x]` variable
- `[x, x, x, x, 0, 0, ..., x, 1, x]` template
- `[1, 2, 4, x, 0, 0, ..., x, x, x]` template
- `[x, x, x, x, 1, 2, ..., x, x, x]` variable
- `[x, x, x, x, 0, 0, ..., x, 1, x]` template
- `[x, x, x, x, 1, 2, ..., x, x, x]` variable
- `[x, x, x, x, 0, 0, ..., x, 1, x]` template
- `...`

We can represent any word even unseen words
Word classifier

- Train **supervised** machine learning classifier
  - E.g., SVM, Random forest.
- The whole framework of LogParse is unsupervised
  - We used unsupervised methods to generated training set

Character count vector:
```
Character count vector:  
a - z, A - Z, 0 - 10, symbol  label
[1, 2, 4, x, 0, 0, ..., x, x, x]              template
[x, x, x, x, 1, 2, ..., x, x, x]              variable
[x, x, x, x, 0, 0, ..., x, 1, x]              template
[1, 2, 4, x, 0, 0, ..., x, x, x]              template
[x, x, x, x, 1, 2, ..., x, x, x]              variable
[x, x, x, x, 0, 0, ..., x, 1, x]              template
[x, x, x, x, 1, 2, ..., x, x, x]              variable
[x, x, x, x, 0, 0, ..., x, 1, x]              template
...```

The whole framework is still unsupervised
Online log parsing

1. New templates generation
2. Templates matching
New templates generation

- **Steps:**
  - Classify each word by the trained word classifier.
  - Construct a new template by combining all template words.

---

**Template set:**

<table>
<thead>
<tr>
<th>Template</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>T&lt;sub&gt;1&lt;/sub&gt;</td>
<td>Interface * changed state to down</td>
</tr>
<tr>
<td>T&lt;sub&gt;2&lt;/sub&gt;</td>
<td>Vlan-interface * changed state to down</td>
</tr>
<tr>
<td>T&lt;sub&gt;3&lt;/sub&gt;</td>
<td>Interface * changed state to up</td>
</tr>
<tr>
<td>T&lt;sub&gt;4&lt;/sub&gt;</td>
<td>Interface * changed state to down</td>
</tr>
</tbody>
</table>

**Template match:**

<table>
<thead>
<tr>
<th>Historical logs</th>
<th>Template match</th>
</tr>
</thead>
<tbody>
<tr>
<td>L&lt;sub&gt;1&lt;/sub&gt; - Interface ae3, changed state to down</td>
<td>L&lt;sub&gt;1&lt;/sub&gt; - T&lt;sub&gt;1&lt;/sub&gt;, ae3</td>
</tr>
<tr>
<td>L&lt;sub&gt;2&lt;/sub&gt; - Vlan-interface vl22, changed state to down</td>
<td>L&lt;sub&gt;2&lt;/sub&gt; - T&lt;sub&gt;2&lt;/sub&gt;, vl22</td>
</tr>
<tr>
<td>L&lt;sub&gt;3&lt;/sub&gt; - Interface ae3, changed state to up</td>
<td>L&lt;sub&gt;3&lt;/sub&gt; - T&lt;sub&gt;3&lt;/sub&gt;, ae3</td>
</tr>
<tr>
<td>L&lt;sub&gt;4&lt;/sub&gt; - Interface ae1, changed state to down</td>
<td>L&lt;sub&gt;4&lt;/sub&gt; - T&lt;sub&gt;4&lt;/sub&gt;, ae1</td>
</tr>
</tbody>
</table>

---

**Historical logs:**

- L<sub>1</sub>. Interface ae3, changed state to down
- L<sub>2</sub>. Vlan-interface vl22, changed state to down
- L<sub>3</sub>. Interface ae3, changed state to up
- L<sub>4</sub>. Interface ae1, changed state to down

**Online logs (new type):**

- L<sub>5</sub>. Vlan-interface vl22, changed state to up

---

**Log parsing problem → word classification problem**

---

2021/1/27

Weibin Meng
Template matching

- Build a prefix-tree for template matching
- Each root-to-leaf path is a template

Templates:

- $T_1$. Interface *, changed state to down
- $T_2$. Vlan-interface *, changed state to down
- $T_3$. Interface *, changed state to up

Template update:

- $T_4$. Vlan-interface *, changed state to up

Online log:

- Interface ae3, changed state to down
Datasets & Baselines

Datasets:

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Description</th>
<th># of logs</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPC</td>
<td>High performance cluster</td>
<td>433,489</td>
</tr>
<tr>
<td>HDFS</td>
<td>Hadoop distributed file system</td>
<td>11,175,629</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>ZooKeeper service</td>
<td>74,380</td>
</tr>
<tr>
<td>Hadoop</td>
<td>Hadoop MapReduce job</td>
<td>394,308</td>
</tr>
</tbody>
</table>

Baselines:

- **Drain** (ICWS’17), **FT-tree** (IWQoS’18) which claimed to support template update
- **LogSig** (CIKM’11), **Spell** (ICDM’16), **IPLoM** (KDD’09) don’t support template update
Evaluation on Intra-service adaptiveness

Results of baselines when all the logs are used for training

All baselines perform good in offline stage

Results of baselines when only 10% of logs are used for training

All baselines perform bad for online matching and update

Results of LogParse when only 10% of logs are used for training

Accuracy of LogParse is even higher than baselines trained by all logs.
Evaluation on stability

LogParse is stable to the scale of training data.

Template accuracy as the percentage of training data increases from 10% to 90%.
Evaluation on cross-service adaptive

<table>
<thead>
<tr>
<th>Training data (service A)</th>
<th>Testing data (service B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HPC</td>
</tr>
<tr>
<td>HPC</td>
<td>-</td>
</tr>
<tr>
<td>HDFS</td>
<td>0.982</td>
</tr>
<tr>
<td>ZooKeeper</td>
<td>0.993</td>
</tr>
<tr>
<td>Hadoop</td>
<td>0.983</td>
</tr>
</tbody>
</table>

On average, LogParse achieves a cross-service accuracy of 0.980
### Evaluation on compression

<table>
<thead>
<tr>
<th>Type</th>
<th>Method</th>
<th>HPC</th>
<th>HDFS</th>
<th>Zookeeper</th>
<th>Hadoop</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shore-term storage</td>
<td>LogParse</td>
<td>13.0%</td>
<td>14.0%</td>
<td>19.6%</td>
<td>4.6%</td>
<td>12.8</td>
</tr>
<tr>
<td></td>
<td>bzip</td>
<td>9.6%</td>
<td>17.4%</td>
<td>9.7%</td>
<td>6.4%</td>
<td>10.8%</td>
</tr>
<tr>
<td></td>
<td>7zip</td>
<td>9.7%</td>
<td>18.1%</td>
<td>9.4%</td>
<td>5.9%</td>
<td>10.8%</td>
</tr>
<tr>
<td></td>
<td>zip</td>
<td>11.4%</td>
<td>20.9%</td>
<td>10.1%</td>
<td>7.2%</td>
<td>12.4%</td>
</tr>
<tr>
<td>Long-term storage</td>
<td>LogParse+bzip</td>
<td>1.4%</td>
<td>2.1%</td>
<td>2.4%</td>
<td>1.2%</td>
<td>1.8%</td>
</tr>
<tr>
<td></td>
<td>LogParse+7zip</td>
<td>2.3%</td>
<td>2.6%</td>
<td>2.8%</td>
<td>1.7%</td>
<td>2.3%</td>
</tr>
<tr>
<td></td>
<td>LogParse+zip</td>
<td>2.2%</td>
<td>2.6%</td>
<td>2.8%</td>
<td>1.7%</td>
<td>2.3%</td>
</tr>
</tbody>
</table>

LogParse is helpful to log compression
Conclusion

LogParse, an adaptive log parsing method
- Intra-service
- Cross-service

Log compression, an application of LogParse
- Assign template for any given log

An open-source toolkit
THANKS

Q&A

mwb16@mails.tsinghua.edu.cn

Toolkit: https://github.com/WeibinMeng/LogParse