Extracting Log Patterns from System Logs in LARGE

Yining Zhao

Computer Network Information Center
Chinese Academy of Sciences
HPBDC, 27 May 2016
Content

- ScGrid in CAS
- The System of LARGE
- Log Patterns – Why?
- 2 Algorithms and Optimizations of Extracting
  - Identical Word Rate
  - Tree-matching
  - Optimizations
- Comparisons on Performances
- Conclusion
ScGrid in CAS

- Scientific Computing Grid Environment
- Integrated by many supercomputing centers in China
  - CNIC the head center
  - 8 national centers
  - 18 institute centers
- Using SCE middleware developed by SCCAS
- Provide computing resources to researches in various fields
  - Meteorology, Metal Forging,
  Fluid Mechanics, High Energy Physics,
  Computational Chemistry, Astrology…
The System of LARGE

- Log Analyzing fRamework in Grid Environments
- Processing logs produced by the environment
  - gathering logs
  - processing, doing statistics and analyzing
  - producing feedbacks
- Two major types of logs
  - system logs – by log service in operating systems
  - SCE logs – by SCE middleware and job scheduling processes
- Helping the environment run correctly and steadily
  - generate alerts for particular patterns of logs
  - provide data for system analysis and maintenance
    - user behavior report
    - system errors and faults
The System of LARGE
Log Patterns – Why?

- We want to be alerted for logs in certain patterns, but…
  - too many logs for human to read
  - need to summarize patterns before defining alert rules
- Set of log patterns in our context:
  - patterns are different from each other
  - covering all logs in original set
  - significantly less than original
- The process of using log patterns
  - filter and remove frequent normal logs
  - use *algorithms of extracting log patterns* to get the set of patterns
  - manually check the set and pick out abnormal patterns
  - define rules to generate alerts for these patterns
Algorithm of IWR

• Algorithm of identical word rate – a straight forward way
  – identical words
    • 2 words that are identical
    • and in the same position in 2 original logs
  – identical word rate: \( \frac{\text{number of identical words}}{\text{total words}} \)
  – predefined threshold \( t \)

• If IWR is greater than \( t \), the two logs are in one pattern
• Logs with different length has IWR of ZERO!

It is a good day
It isn’t a bad day

\[ t = 0.66 \]
\[ \text{IWR} = 60\% \]
NOT in the same pattern!
Algorithm of IWR

• Process of algorithm of IWR
  – set threshold \( t \) and initial empty pattern set \( P \)
  – for each new incoming logs, compute IWR with each pattern in \( P \)
  – if pattern matched, skip to next; if none matched, add to \( P \)

• \( P \) will be affected by order of input
  – 3 logs, \( L1:\{a, b, c\} \) \( L2:\{a, b, d\} \) \( L3:\{d, b, c\}, t = 0.6 \)
  – in order of \( l1, l2, l3 \), only 1 pattern left
  – in order of \( l2, l1, l3 \), 2 patterns left

• Not ideal in complexity
  – \( O(n^2) \)
Algorithm of Tree-matching

- Different in storing and matching structure with IWR
- Words stored orderly in a tree
  - branches for different words
- When performing the algorithm
  - compare each word in the incoming log from root
  - if successfully matched to leaf, check next
  - if unmatched found, create a new branch
- To get pattern set, use depth-first traverse
- Better complexity, but worse result
  - $O(n \log(n))$
  - unacceptable number of patterns
  - need optimizations
Algorithm Optimizations

• Tree pruning
  – if two nodes have the same subtrees, merge to a key node (<>)
  – after previous step, merge all key nodes to keep uniqueness
  – key node can be matched to any word, but only the last option

• we can do this because...

Login failed from user alice: password failed
Login failed from user bob: password failed
Login failed from user chris: password failed

key information position
Algorithm Optimizations

• Word converting function
  – a preprocessing optimization
  – converting commonly appeared expressions to predefined strings

• We can do this because…
  – At this stage, IPs, usernames, etc. are not essential
  – could be a distraction for extracting and enlarge pattern set

• Helpful for tree-matching, but not IWR
  – IWR has higher tolerance for differences of words
  – Tree-matching is more sensitive

<table>
<thead>
<tr>
<th>Word Format in Regular Expression</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0-9]+</td>
<td>&lt;NUMBER&gt;</td>
</tr>
<tr>
<td>name=[A-Za-z][A-Za-z0-9_]+</td>
<td>name=&lt;USER&gt;</td>
</tr>
<tr>
<td>UID=[0-9]+</td>
<td>UID=&lt;NUMBER&gt;</td>
</tr>
<tr>
<td>GID=[0-9]+</td>
<td>GID=&lt;NUMBER&gt;</td>
</tr>
</tbody>
</table>
Comparisons on Performances

original: 49079 input logs

A. IWR, t = 0.66
B. IWR, t = 0.6
C. IWR + word converting, t = 0.66
D. IWR + word converting, t = 0.6
E. Tree-matching
F. Tree-matching + pruning
G. Tree-maching + word converting
H. Tree-matching + both
Comparisons on Performances

Tree-matching has significantly reduced the number of comparisons

A. IWR, t = 0.66
B. IWR, t = 0.6
C. IWR + word converting, t = 0.66
D. IWR + word converting, t = 0.6
E. Tree-matching
F. Tree-matching + pruning
G. Tree-matching + word converting
H. Tree-matching + both

Word converting function further reduces the number of comparisons
Comparisons on Performances

Word converting function is quite time costly

A. IWR, $t = 0.66$
B. IWR, $t = 0.6$
C. IWR + word converting, $t = 0.66$
D. IWR + word converting, $t = 0.6$
E. Tree-matching
F. Tree-matching + pruning
G. Tree-matching + word converting
H. Tree-matching + both
Comparisons on Performances

Extracting efficiency (time and comparison costs): tree-matching is better

A. Original
B. IWR, t = 0.6
C. IWR + word converting, t = 0.66
D. IWR + word converting, t = 0.6
E. Tree-matching
F. Tree-matching + pruning
G. Tree-matching + word converting
H. Extracting effect (number of patterns): IWR is better
Comparisons on Performances

Lower threshold gives lesser patterns in IWR

<table>
<thead>
<tr>
<th></th>
<th>A. IWR, t = 0.66</th>
<th>B. IWR, t = 0.6</th>
<th>C. IWR + word converting, t = 0.66</th>
<th>D. IWR + word converting, t = 0.6</th>
<th>E. Tree-matching</th>
<th>F. Tree-matching + pruning</th>
<th>G. Tree-matching + word converting</th>
<th>H. Tree-matching + both</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of Comparisons</td>
<td>3395619</td>
<td>2823256</td>
<td>2279185</td>
<td>2251530</td>
<td>1270695</td>
<td>1310259</td>
<td>324131</td>
<td>331222</td>
</tr>
<tr>
<td>Time Cost</td>
<td>967</td>
<td>968</td>
<td>2304</td>
<td>2298</td>
<td>488</td>
<td>530</td>
<td>572</td>
<td>601</td>
</tr>
<tr>
<td>No. of Log Patterns</td>
<td>124</td>
<td>93</td>
<td>78</td>
<td>74</td>
<td>275</td>
<td>655</td>
<td>103</td>
<td>16</td>
</tr>
</tbody>
</table>
Comparisons on Performances

Number of comparisons has nearly direct proportion to the number of extracted patterns

A. IWR, $t = 0.66$
B. IWR, $t = 0.6$
C. IWR + word converting, $t = 0.66$
D. IWR + word converting, $t = 0.6$
E. Tree-matching
F. Tree-matching + pruning
G. Tree-maching + word converting
H. Tree-matching + both
Comparisons on Performances

- Pure tree-matching:
  - good in extract efficiency
  - very bad in extracting effect
- A. IWR, $t = 0.6$
- B. IWR + word converting, $t = 0.66$
- C. IWR + word converting, $t = 0.6$
- D. Tree-matching
- E. Tree-matching + pruning
- F. Tree-matching + word converting
- G. Tree-matching + word converting
- H. Tree-matching + both

![Diagram showing comparisons](chart.png)

- No. of Comparisons
- Time Cost
- No. of Log Patterns
Comparisons on Performances

Pruning has greatly improved the extracting effect

A. IWR, $t = 0.66$
B. IWR, $t = 0.6$
C. IWR + word converting, $t = 0.66$
D. IWR + word converting, $t = 0.6$
E. Tree-matching
F. Tree-matching + pruning
G. Tree-matching + word converting
H. Tree-matching + both
Comparisons on Performances

original: 49079 input logs

Tree-matching with both optimizations: good in extracting efficiency and effect, satisfying algorithm!
A. IWR, $t = 0.6$
B. IWR, $t = 0.6$
C. IWR + word converting, $t = 0.6$
D. IWR + word converting, $t = 0.6$
E. Tree-matching
F. Tree-matching + pruning
G. Tree-maching + word converting
H. Tree-matching + both
Conclusion

• LARGE is a log analyzing system
• When monitoring system logs, we need to extract log patterns
• Two algorithms: IWR and tree-matching, plus optimizations
  – tree-matching with two optimizations looks good

• Future work
  – what if more than one key position in tree-matching algorithm?
  – we may use parallel computing to accelerate log processing