Mining Dependency in Distributed Systems through Unstructured Logs Analysis
Qiang FU
Microsoft Research Asia
Motivation

- There are a lot of dependency relations in distributed systems
- Dependencies are essential to trace problems and locate root error position

Problems:
- Large amount of log data from large scale systems
- Increasing complexity of systems
- Manually constructing dependency relations is time consuming and tedious
- Administrators are often lack of system knowledge

Automatic dependency mining tool is desired
Background

- Component dependencies are causal relations between system actions or events of different components.
- Logs usually record important system actions or events for trouble shooting, and a log sequence reflects an execution path of a distributed program.
- Dependencies between log messages can imply inter-component dependencies.
- Therefore, the inter-component dependency can be learned from the log sequences.
Basic idea

- We have three observations for mining dependency
  - Co-occurrence: if B depends on A, then B is likely to occur within a short interval after A’s occurrence
  - Parameter correspondence: two dependent log messages often contain at least one identical parameter
  - Delay consistency: the delay’s variance of dependent logs is often much smaller than that of independent logs
Challenges & Assumptions

- **Challenges**
  - Log messages are often unstructured free form text strings and not easy to be understood by a computer program.
  - No enough documents to describe the detail of the log formats.
  - Machines are not precisely synchronized, and time difference between machines may cause disorder of log messages.

- **Assumptions**
  - Each log message has a timestamp
  - Machines in the system are roughly synchronized (< 200ms, NTP)
Approach overview

- Learning Dependency Graph

- Locating root error
Log message parsing

Problem
- console logs are often unstructured free form text strings

How to structure it?
- Log message -> log key + parameters
- Log key is the constant part in a log-print statement
- Parameter is the value of printed variable

New job added to schedule, jobId = 8821, priority = 64
Log message parsing

Basic idea
- log messages printed by the same log-print statement usually have high similarity
- Log key is constant and parameters have many different values
- We group log messages printed by the same log-print statement together to extract log key and parameter.

Result
- Each log message is expressed as a tuple (log key, param1, param2,...)

Algorithm
- Erasing contents satisfying typical parameter patterns
- Clustering raw log keys into initial groups
- Splitting initial groups into final groups
- Extract common part as log key
Example of log key extraction

Log Message 1: [172.23.67.0:4635] TCP Job name UpdateIndex
Log Message 2: [172.23.67.0:4635] TCP Job name DropTable
Log Message 3: [172.23.67.0:4635] TCP Job name UpdateTable
Log Message 4: [172.23.67.0:4635] TCP Job name DeleteData
Log Message 5: Image file of size 57717 loaded in 0 seconds.
Log Message 6: Image file of size 70795 saved in 0 seconds.
Log Message 7: Edits file \tmp\hadoop-Rico\dfs\name\current\edits
of size 1049092 edits # 2057 loaded in 0 seconds.

Erasing parameters by empirical rules

Raw log key 1: [] TCP Job name UpdateIndex
Raw log key 2: [] TCP Job name DropTable
Raw log key 3: [] TCP Job name UpdateTable
Raw log key 4: [] TCP Job name DeleteData
Raw log key 5: Image file of size loaded in seconds.
Raw log key 6: Image file of size saved in seconds.
Raw log key 7: Edits file of size edits # loaded in seconds.

Splitting initial groups

Raw log key 1: [] TCP Job name UpdateIndex
Raw log key 2: [] TCP Job name DropTable
Raw log key 3: [] TCP Job name UpdateTable
Raw log key 4: [] TCP Job name DeleteData
Raw log key 5: Image file of size loaded in seconds.
Raw log key 6: Image file of size saved in seconds.
Raw log key 7: Edits file of size edits # loaded in seconds.

Clustering raw log keys

Raw log key 1: [] TCP Job name UpdateIndex
Raw log key 2: [] TCP Job name DropTable
Raw log key 3: [] TCP Job name UpdateTable
Raw log key 4: [] TCP Job name DeleteData
Raw log key 5: Image file of size loaded in seconds.
Raw log key 6: Image file of size saved in seconds.
Raw log key 7: Edits file of size edits # loaded in seconds.

Initial Group 1

Initial Group 2

Extracting log keys

log key 1: [] TCP Job name
log key 2: Image file of size loaded in seconds.
log key 3: Image file of size saved in seconds.
log key 4: Edits file of size edits # loaded in seconds.
Dependency mining

- Dependency we mined are about log keys
  - Each log key corresponds to a specific log-print statement in the source code
  - Dependencies of log keys reflect dependencies of execution logic
    - They also reflect dependencies of different classes of system actions or events
- The log messages are instances of their corresponding log keys
Two steps for mining dependency

- Dependency = dependent log key pair + dependency direction
- The time difference of machines may lead to disorder of log messages
- Hard to determine the dependent log key pair and its direction simultaneously
  - Find dependent log key pairs
  - Determine their dependency direction
Significance of dependent log key pair

- How to measure significance of dependent log key pair
- The conditional probability that encodes the co-occurrence and parameter correspondence
  - A parameter correspondence pattern is defined as a quadruple: $Q = (s, d_1, q, d_2)$; $s, q$ are log keys; $d_1, d_2$ are positions
  - Account the fraction of instances that satisfy the defined correspondence pattern
Calculate conditional probability

- \[ P(Q|s) = \frac{C_s(Q)}{O(s)} \quad //\text{note: } Q = (s, d_1, q, d_2) \]
  \- \( O(s) \) is the number of messages whose log key is \( s \)
  \- \( C_s(Q) \) is the number of messages \( A \) that satisfy:
    - \( K(A)=s; \)
    - there is at least one message \( B \) satisfying that \( K(B)=q, |T(A)-T(B)|<t_d, \) and \( PV(A,d_1)=PV(B,d_2) \)

- Similarly, \( P(Q|q) = \frac{C_q(Q)}{O(q)} \)

- \( s \) and \( q \) are dependent, if there is a \( Q \) satisfy
  \[ \max(P(Q|s), P(Q|q)) \geq Th_{cp} \]
Reduce the computational complexity

- $N$ log keys, each with $M$ parameters, there will be $N(N-1)M^2$ quadruples

Reduce the computational complexity

- We only estimate inter-component dependencies
  - they are more interested in system management and fault localization

- We filter out log key pairs by preprocessing
  - If conditional probability of log key pair $(s, q)$ are smaller than $Th_{cp}$, then all correspondence patterns of $q$ and $s$ can be ignored because they must smaller than $Th_{cp}$ too
Variance based false positive reduction

- Some routine messages can satisfy our significance dependency measurement
- It leads to false positive detection
- Such false positive can be removed by delay variance analysis
  - Two log keys of a false positive co-occur within the time window by chance
  - The time delay between them are often quite random
  - The delay variances of independent log key pairs are much bigger than that of dependent log key pairs
Determine the dependency direction (1)

- The later one depends on the earlier one
- If machines’ clock are precisely synchronized, the direction is easy to determine
- The clock difference may lead to messages’ disorder
- For each dependent log key pair, we account time delays of all their instances, and decide the dependency direction according to Bayesian decision theory
Determine the dependency direction (2)

- For a dependent log key pair \((s, q)\), we can find \(n\) log message samples \((s_i, q_i)\), \(i=1,...n\)
- \(T = t + \delta\), \(t\) is the local time, \(T\) is the absolute time, \(\delta\) is the time deviation
- If \(\delta_{s_i}\) and \(\delta_{q_i}\) \((i = 1,...n)\) are i.i.d., then the true time delay is asymptotically conforms to a normal distribution whose mean is equal to
  \[
  \mu_{sq} = \frac{\sum_{i=1}^{n} (t_{s_i} - t_{q_i})}{n}
  \]
- If \(\mu_{sq} < 0\), \(q\) depends on \(s\); else \(s\) depends on \(q\)
Dependency Pruning

- For a dependent log key pair, we may also detect dependencies among their neighbors.
- We need to remove the indirect dependency relations to get compact and clear dependency graph.
- Example
  - D2 and D3 are redundant dependencies because they can be inferred from D1 and D4 respectively.
Experiments

◦ Test bed
  ◦ Hadoop(v0.19) is a well-known open source implementation of Map-Reduce and GFS
  ◦ 16 machines, one is for NameNode and JobTracker, others are DataNode and TaskTracker
  ◦ 10 word count jobs, each text file is 10 G

<table>
<thead>
<tr>
<th>Machine</th>
<th>Basic configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>PT01</td>
<td>Intel quad-core <a href="mailto:E5410@2.33G">E5410@2.33G</a>, 8G RAM</td>
</tr>
<tr>
<td>PT03~PT05</td>
<td>Intel dual-core <a href="mailto:E3110@3.0G">E3110@3.0G</a>, 8G RAM</td>
</tr>
<tr>
<td>PT06~PT11</td>
<td>Intel quad-core <a href="mailto:E5430@2.66G">E5430@2.66G</a>, 8G RAM</td>
</tr>
<tr>
<td>PT12~PT17</td>
<td>AMD Quad-Core <a href="mailto:2376@2.29G">2376@2.29G</a>, 8G RAM</td>
</tr>
</tbody>
</table>
Ground truth

- We use loose parameter to obtain about 150 candidate dependencies
  - time window as 3 seconds
  - probability threshold as 0.5
- We manually check those candidate dependencies as our ground truth
Experiments (1)

- $\tau$ vary from 0.1 to 3 seconds
- $\tau$ around 1 second is a good parameter
- Without variance based false positive reduction
Experiments (2)

- Probability threshold increases from 0.5 to 0.95
- Refinement is the variance based false positive reduction
- $0.8 \leq Th_{cp} \leq 0.9$ is the good choice of parameter
Dependency example

- A new task added in JobTracker causes a new task lunched by a TaskTracker.
- A Reduce task starts its commit action, then the NameNode performs a data block allocation operation, and the DataNode carries out a HDFS_WRITE operation.
Root Error Localization (1)

- An error may propagate to other components due to inter-component dependencies.
- A group of related errors from different components are often detected at the same time.
- How to find the root error from a group of related errors?
Root Error Localization (2)

Basic idea

1. There are two detected errors which make $q_2$ and $s_3$ as inaccessible states.
2. If $s_3$ depends on $q_2$, it is reasonable to conclude that Error 1 causes Error 2.
Summary

- We propose a dependency mining algorithm to discover inter-component dependencies from console log data.

- Advantages
  - Mine dependency in black-box manner without application specific knowledge
  - Require neither system instrumentation nor annotation
  - Easy to apply in the wide range of systems
Thanks!

Q & A