Workflow motif: an abstraction for debugging distributed systems

Mania Abdi, Ata Turk, Rich Megginson, Peter Desnoyers, Mark Crovella, Raja Sambasivan
How we build complex distributed systems

- We use powerful abstractions to build distributed systems
  - Build complex systems from simpler building blocks
  - Hide details of building blocks' implementation.
How we build complex distributed systems

• We use powerful abstractions to build distributed systems
  • Build complex systems from simpler building blocks
  • Hide details of building blocks' implementation.
How we debug systems

• Defacto approach to debug is to use less sophisticated abstractions

• Sometimes no abstractions whatsoever, such as raw logs

Extreme mismatch between building systems vs debugging them: key reason why diagnosis is challenging
Toward better diagnosis abstractions

- Traces show repetitive patterns within and across many requests.
  - Pattern are repetitive building blocks of how distributed system behave
  - Extracting patterns and understanding their characteristics is very powerful.
Workflow Motifs: A Novel Diagnosis Abstraction

- Graphs that describe frequent processing patterns in the workflow of how requests are processed along with their performance.

- Building blocks of distributed systems runtime behavior.
  - E.g., Work done to write data to storage node,
  - E.g., Work done to elect leader in a consensus protocol.

- Mitigates complexity during diagnosis by:
  - Allowing problems to be understood in terms of behavioral building blocks.
  - Allow the details irrelevant to a given problem to be hidden.
Features of workflow motifs

- Motif two is repeated not only across requests of the same type but also many different types of requests.
- Workflow motifs are hierarchical
- Workflow motifs are composable
- Workflow motifs consist of performance characteristics
Key enabler: Workflow centric tracing

- Captures how each request is processed within and among different components of distributed systems.
- This is enabled by Propagates unique ID with each request as it is executed by the system

Diagram:
- Client Application
- Table Store
- Server
- Backend Storage
- Read: table A, 1GB
Approach: Frequent subgraph mining

• Find frequent subgraph of different sizes,
  • Expand frequent subgraph by one node and determine if larger subgraph is still frequent
• A frequent subgraph frequency is more than a user defined threshold (support level).
• Used in other domains:
  • Biology for DNA matching,
  • chemistry for component matching
Workflow motif architecture

- Motif query engine
- Trace database
- Workflow motif identification
- Trace annotation engine

- Context propagation
- Trace point
- Sampling
- Trace construction

- Distributed Application
- Client
- Metadata
- Storage Service

Contrast execution
Identify anomalies
Optimizations
Application comprehension

Identify anomalies
Optimizations
Application comprehension

Identify anomalies
Optimizations
Application comprehension

Identify anomalies
Optimizations
Application comprehension

Identify anomalies
Optimizations
Application comprehension

Identify anomalies
Optimizations
Application comprehension
Outline

• Introduction
• Design
• **Requirements**
• Use cases
R1: Motifs must preserve work order

- Motifs should preserve ordering of works as it represented in the traces.
  - Correct execution in lower stack layer
  - Identify implementation bugs in the code
  - Identify performance problem
- They should preserve application semantic
R2: Motifs are frequent across and within traces

• We should capture workflow motifs that occur many times within a small set of request workflows, but will not occur in a large set of them.

• We should also capture workflow motifs that occur only a few times within individual workflows, but occur repeatedly in many different ones.
R3: Motifs show performance characteristics

• To be useful for performance debugging, workflow motif must show the performance characteristics of the processing pattern:
  • Distribution of overall execution time,
  • The latency of the critical path,
  • The distribution of more detailed timing.
# Comparison of different graph mining tools

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Approach</th>
<th>Edge Order</th>
<th>In/across graphs</th>
<th>Scalable</th>
</tr>
</thead>
<tbody>
<tr>
<td>FSG</td>
<td>Joint-base</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Gaston</td>
<td>Pattern growth</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Grami</td>
<td>CSP</td>
<td>Yes</td>
<td>Yes</td>
<td>Optimization</td>
</tr>
<tr>
<td>Arabesque</td>
<td>Pattern growth</td>
<td>No</td>
<td>Yes</td>
<td>Distributed</td>
</tr>
<tr>
<td>ASAP</td>
<td>Pattern growth and sampling</td>
<td>No</td>
<td>Yes</td>
<td>Distributed</td>
</tr>
</tbody>
</table>
Examples of workflow motifs

• An example of workflow motif generated by Gaston, for a set of HDFS traces

Read from network motif

Read block Motif
Outline

• Introduction
• Design
• Requirements
• Use cases
Use Cases

• Contrast and compare request workflows
  • Finding common motifs within each execution and compare their structure.
• Improve Slow performance
  • Identify slowest motifs and present them to engineers so that they can optimize them.
• Identify anomaly
  • Identify request containing motifs that usually don’t occur together
Summary

• We use powerful abstractions to build distributed systems,
• De-facto approach to debug is to use less sophisticated abstractions,
• We introduce workflow motif, a novel abstractions for debugging and diagnosis distributed system,
  • A workflow motif is a frequent processing pattern that could be extracted from workflow traces
  • Workflow centric tracing is key enabler
• Our approach is to use frequent subgraph mining tools.
  • Arabesque, Grami and Gaston features are closest toWorkflow motif requirements.