The workflow motif: a powerful abstraction for debugging distributed systems

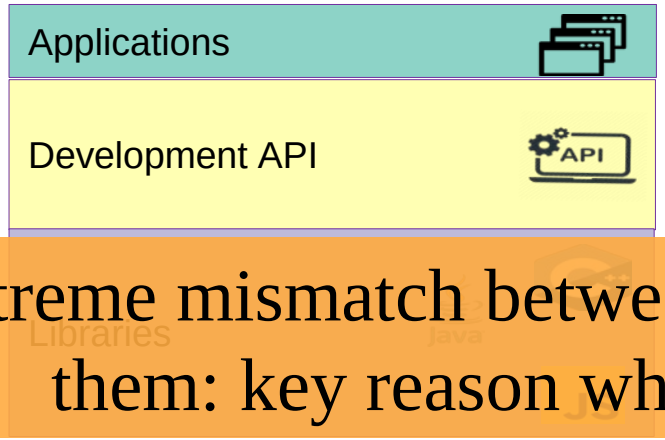
Mania Abdi, Golsana Ghaemi, Mark Crovella, Orran Krieger, Peter Desnoyers, Ata Turk, Raja Sambasivan

Northeastern

BU

Abstractions for Building vs. Debugging Production Systems

- Abstractions enable developers to build complex distributed systems
  - **Build complex systems from simpler building blocks**
  - **Hide details of implementation.**

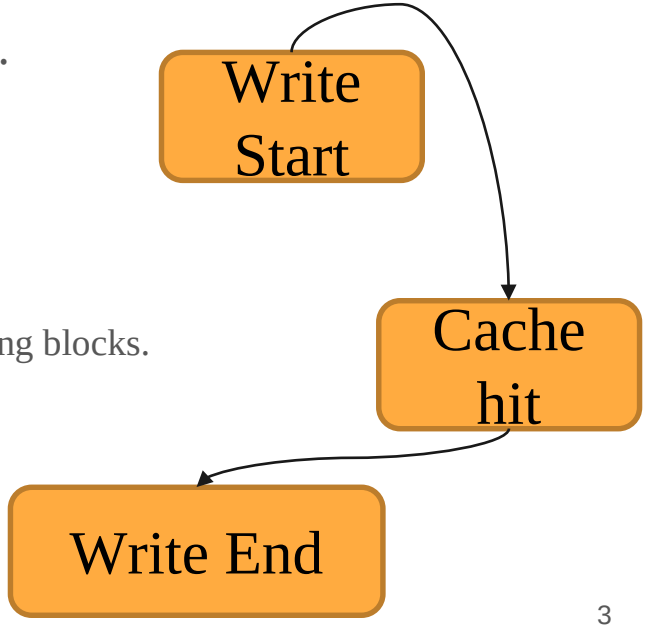


- De facto approach to debugging is to use no abstractions, whatsoever.
  - **i.e. Logs of low level events**

Extreme mismatch between building systems vs debugging them: key reason why diagnosis is so challenging

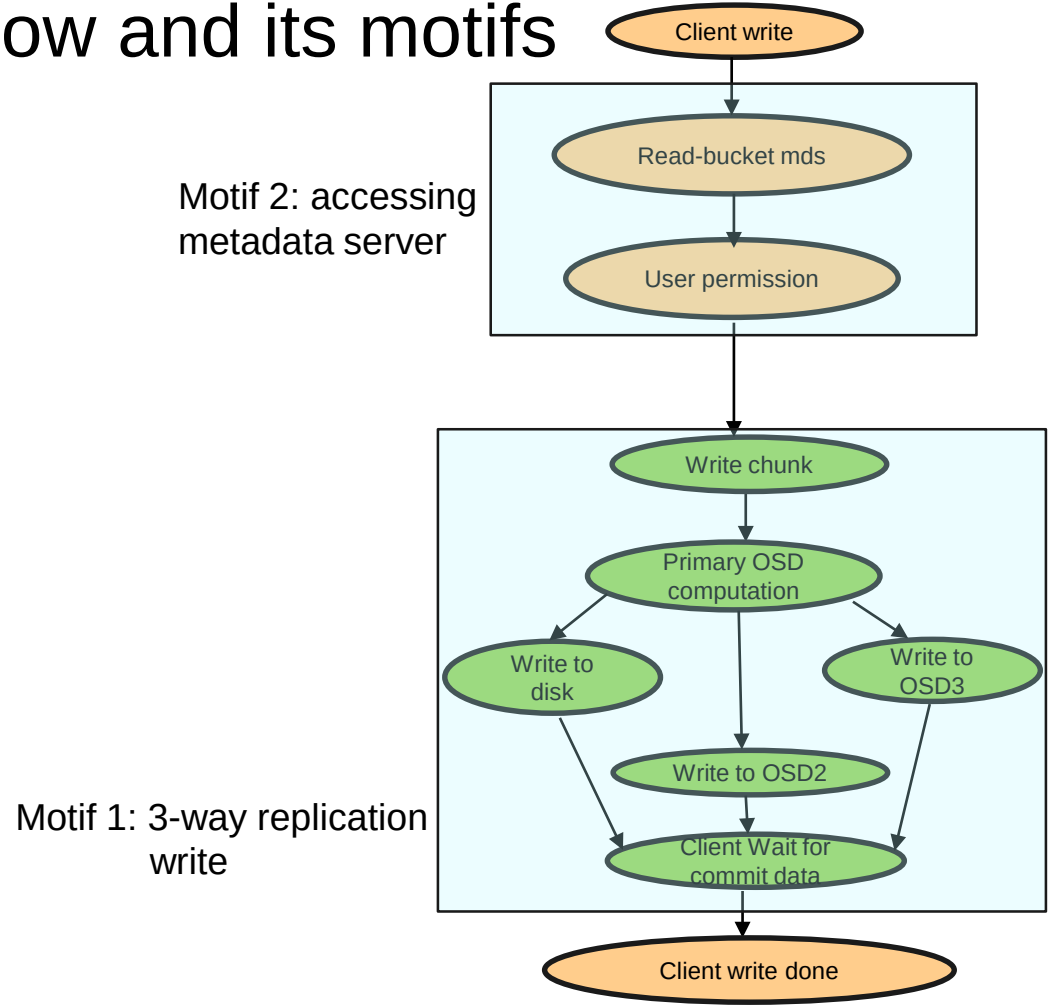
Workflow Motifs: A Novel Diagnosis Abstraction

- 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.



Example of write workflow and its motifs

- Simplified write request workflow consists of work done in client and storage nodes
- Motif one is repeated among many write requests
- Motif two is repeated not only across write requests but also many different types of requests



Use Cases

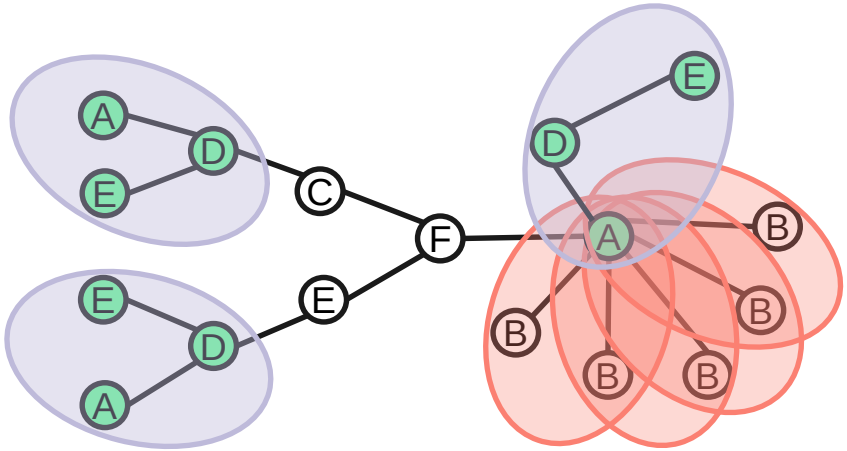
- Contrast distributed application execution
  - Fining common motifs with in each execution and compare their structure.
- Improve Slow performance
  - Identify slowest motifs and present them to engineers so that they can optimize them.
- Flag anomalies
  - Identify request containing motifs that usually don’t occur together
- Reveal emergent behaviors
  - Identify patterns that usually don’t occur together.

Key Enabler: End-to-End tracing

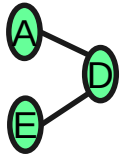
- Captures how each request is processed within and among different components of distributed systems
- How it works:
  - Propagates unique ID with each request as it is executed by the system
  - Executed log points are stored to disk and tagged with unique ID
  - Creates traces by stitching together log points with the same ID
  - Traces are which are Directed Acyclic Graphs (DAG) :
    - Nodes are trace points
    - Edges represent the causal relationships and also latency between nodes

Approach: mine traces using subgraph mining algs

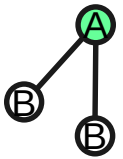
- Used in Biology for DNA matching, chemistry for component matching.
  - Example: gaston, pafi
- Way they work:
  - Find smallest frequent subgraph
  - Expand frequent subgraph by one node and determine if larger subgraph is still frequent



- Pattern 1:



- Pattern 2:



Progress So Far

- Added end-to-end tracing to a distributed storage application, CEPH
- Exploring different subgraph mining algorithms by using them extract subgraphs.

Key Research Questions

- How should we modify the frequent subgraph mining algorithms to suit our domain specific needs?
- What other approaches can be used to identify motifs?
- What properties tracing infrastructure should support to capture motifs?
- What other use cases motifs can be useful for?