Transparent System Call Based Performance Debugging for Cloud Computing

Nikhil Khadke, Michael P. Kasick, Soila P. Kavulya, Jiaqi Tan, and Priya Narasimhan,

PARALLEL DATA LABORATORY
Carnegie Mellon University
Introduction

• Automated problem diagnosis is challenging
  • Complex system interactions: Concurrency and distributed interactions can obscure root-cause
  • Scale: Large volume of monitoring data
  • Production systems: Lack luxury to modify instrumentation in legacy systems

• Focus of talk
  • Explore feasibility of system calls as transparent instrumentation source for debugging of performance problems
Related Work: Instrumentation

• End-to-end tracing of request flows
  • Instrumentation frameworks [Sambasivan11, Sigelman10]
  • Black-box approach [Aguilera03]
  • Analysis of production logs [Kavulya12, Tan09]

• Performance and event log analysis
  • OS-level logs [Kasick10]
  • Generation of problem signatures [Bodik10, Cohen05]
  • Alarm correlation [Oliner10, Kandula09]
Why System Calls?

• Captures interaction between application & OS
• Rich source of statistical and semantic data
  • Statistical: Disk/network transfer times
  • Semantic: Programs/files accessed, hangs/failures
• Higher reliability than application logs
  • Quality of logs dependent on application developer
  • Logs can become obsolete as system evolves
• Transparency
  • No need to modify to application source code
  • No dependence on hardware architecture
Goals and Assumptions

• Goals
  • Learn profiles of normal behavior using system calls
  • Localize problematic node and type of problem

• Assumptions
  • Majority of nodes exhibit fault-free behavior
  • Workload evenly balanced across nodes
  • Hardware configurations similar across nodes
  • Clocks on nodes are synchronized

• Initial exploration
  • Diagnose resource contention in MapReduce systems
Target System: Hadoop

- Hadoop: Open-source implementation of MapReduce
  - Long running jobs (> 100s): Hard to label failures
  - Large, distributed: Hard to isolate failures
  - Leased cloud infrastructure: Slowdowns are costly
- MapReduce job consists of two main abstractions
  - Maps: Process smaller partition of large job in parallel
  - Reduces: Fetch, merge, and sort output of map tasks
Overview of Approach

Instrument System

• Collect per-node system call traces

Localize Problem

• Identify nodes whose system call profiles differ significantly from peers
• Studied two peer-comparison approaches
System Call Instrumentation

- `strace()`, a UNIX utility captures system calls invoked by program

- We record:
  - Average and total time spent in each system call
  - Number of total and failed system call invocations
  - Percentage of time of spent in the system call
  - Verbose trace of system calls
System Call Scope and Choice

- Network related:
  - accept()
  - connect()
  - bind()
  - socket()

- File System related:
  - access()
  - stat()

- Process related:
  - execve()

- Omitted send/receive calls due to large variance in data processed

- Overhead of naïve implementation: 1.2x
Diagnostic Approach

- Learn per-node profiles of normal behavior

- Two-step threshold for generating alarms
  - **Local alarms:** Accounts for normal variance between nodes (min. and max. bin counts for thresholds)
  - **Global alarms:** Pairwise histogram-comparison flags nodes which differ significantly from peers

- Use heuristics to identify type of problem
Diagnostic Approaches

- **Statistical diagnostic approach**
  - Histogram of invocations of system call per time window

<table>
<thead>
<tr>
<th>% time</th>
<th>seconds</th>
<th>usecs/call</th>
<th>calls</th>
<th>errors</th>
<th>syscall</th>
</tr>
</thead>
<tbody>
<tr>
<td>96.15</td>
<td>0.001524</td>
<td>117</td>
<td>13</td>
<td>7</td>
<td>execve</td>
</tr>
<tr>
<td>3.85</td>
<td>0.000061</td>
<td>0</td>
<td>1705</td>
<td>1198</td>
<td>stat</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>74</td>
<td>44</td>
<td>access</td>
</tr>
<tr>
<td>0.00</td>
<td>0.000000</td>
<td>0</td>
<td>72</td>
<td>1</td>
<td>socket</td>
</tr>
</tbody>
</table>

- **Semantic diagnostic approach**
  - Uses verbose system traces
  - Identify unique system call invocations
    - Unique call → system calls, arguments, return code
    - E.g., `stat(fileX) = -1 ENOENT (No such file or dir)`
Experimental Setup

- Experimental Setup
  - 5 identical machines: Single master and 4 slaves
- Hadoop workloads
  - `wc` – count the frequency of every word in a text corpus of 100,000 words
  - `sort` – sorts 100,000 randomly generated numbers
- Performance problem injection
  - Disk-hog – write 2GB chunks continually to disk
  - Packet-loss – drop 5%, 20% and 50% of the network packets on a node
Results: Terminology

- True positive
  - Faulty node and root-cause correctly identified
- False positive
  - Incorrectly identify a node or provide a wrong root-cause
- Output node set
  - Set of all the nodes that are outputted by our algorithms (and believed to be faulty).
Results: Disk hog

Disk hog signature: More time in stat() + less time in execve()

- True positive rate (Non-speculative)
- True positive rate (Speculative)
- False positive rate (Non-speculative)
- False positive rate (Speculative)

![Bar chart showing statistical and semantic results](chart.png)
Results: Packet Loss

Packet loss signature: More time in connect() + less time in accept()

Results for 50% Packet Loss

- True positive rate (Non-speculative)
- True positive rate (Speculative)
- False positive rate (Non-speculative)
- False positive rate (Speculative)

Statistical
- True positive rate: 0.75
- False positive rate: 0.06

Semantic
- False positive rate: 0.06
Lessons Learned

• Diagnostic approaches
  • Statistical approach more effective than semantic approach
  • Diagnostic accuracy independent of Hadoop workload
  • Speculative execution introduces variance that reduces diagnostic accuracy

• Performance Problems
  • Disk-related issues were easier to detect
  • Lower accuracy for network-related problems probably due to correlated problem manifestation across nodes
Future Work

• How do cope with heterogeneous systems?
  • Normalize behavior across heterogeneous nodes?
  • Limit peer comparison to nearest neighbors?
• How do we reduce instrumentation overhead?
• Can we distinguish between application-level problems and infrastructural problems?
Questions?
Related Work (1)


- **[Cohen05]**: Capturing, indexing, clustering and retrieving system history. Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, Armando Fox. SOSP, 2005.


Related Work (2)

- **[Kavulya12]**: Draco: Statistical Diagnosis of Chronic Problems in Large Distributed Systems. Soila Kavulya, Scott Daniels, Kaustubh Joshi, Matti Hiltunen, Rajeev Gandhi, Priya Narasimhan. DSN 2012


