Problem Diagnosis in Large-Scale Computing Environments

Alexander V. Mirgorodskiy*
VMware, Inc.

Naoya Maruyama†
Department of Mathematical and Computing Sciences
Tokyo Institute of Technology

Barton P. Miller‡
Computer Sciences Department
University of Wisconsin

IEEE/ACM Supercomputing 2006

PURDUE UNIVERSITY

Finding Errors in HPC is Difficult

- Hard analyzing execution of interacting processes
- Bugs in concurrent systems not present in sequential software
- Non-interactive nature complicates error detection

PURDUE UNIVERSITY

Main Contributions

- Locate causes of anomalies in distributed systems
  - Find processes *substantially different* from others
  - Identify the *function* that explains anomalies
- *Automate* problem detection to some extent

Inject Agent to Monitor Processes → **Collect run-time execution data** → **Identifies Anomalies**

Problem Diagnosis Process

Collect control-flow traces until application fails

- Inject agent into every process
- Record function calls/returns

↓

Identify process that fails

- Non-trivial with *silent* or *non-stop failures*
- Identify process that is different from the others
- Distance-based outlier detection
- Can work with or without previous normal traces

↓

Root Cause Identification

- Determine why process behaved differently from the others

Outline

- Fault Model
- Data Collection
- Finding Misbehaving Hosts
- Finding Cause of Anomaly
- Experimental Results

Fault Model

- Non-deterministic fail-stop failures

  If a process crashes, its control flow will stop prematurely
  ➔ its trace will look different than others
- Infinite loops

  Process spends more time in a particular function
- Deadlock, Livelock, Starvation

  Function where process blocks points to location of failure
- Load Imbalance

  Time spent in functions will be different in affected process

Undetectable Problems

- Massive failures

  Problem happens on all nodes

- Problems with no change in the control flow

- Faults that are activated long before its manifestation

  Circular buffer only retains fixed number of recent events

PURDUE UNIVERSITY

Problem Diagnosis Process

Collect control-flow traces until application fails

↓

Identify process that fails

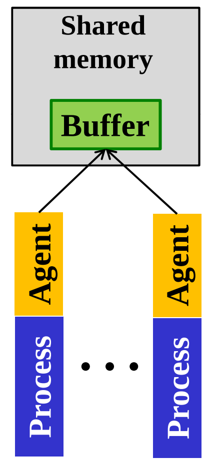
↓

Root Cause Identification

PURDUE UNIVERSITY

Data Collection

- When application starts, an *agent* is injected
  - Agent is a shared library (application's address space)
  - Hijack mechanism to force library loading
  - Records function ***calls*** and ***returns***
  - Published in MMCN '05
- Agents do not communicate to each other
- Buffer saved in shared memory
  - If process dies, buffer is still available



Problem Diagnosis Process

Collect control-flow traces until application fails

↓

Identify process that fails

↓

Root Cause Identification

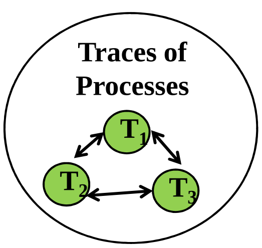
Earliest Last Timestamp

- Process that stopped generating traces is reported as an anomaly

  Effective for **fail-stop problems**

- Simple detection mechanism:
  1. Compare absolute last timestamps (t_i) across hosts
  2. Compute μ (mean) and σ (std. deviation)
  3. If earliest t_i substantially different from μ and σ, report anomaly

Finding Behavioral Outliers



Compute *pair-wise distance*

Assign *Suspect Score*

0.04	0.9	0.06
T_1	T_2	T_3

Dissimilarity between a trace and a collection of normal traces

Rank Traces

0.9 ~ T_2
0.06 ~ T_3
0.04 ~ T_1

Pair-wise Distance Metric

- Distance between traces of two hosts *g* and *h*
- The ***profile*** of a host *h* is a vector ***p***(*h*) of length *F*

$$p(h) = \left(\frac{t(h, f_1)}{T(h)}, \ldots, \frac{t(h, f_F)}{T(h)}\right)$$

F ~ total number of functions in application

i^{th} component is the time $t(h, f_i)$ spent in function f_i

T(h) ~ total runtime of the application $T(h) = \sum_{i=1}^{F} t(h, f_i)$

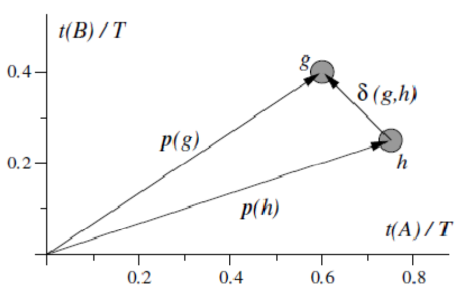
- Can treat different call paths as different functions

(A → B → C) = f_1, (D → E → C) = f_2

Pair-wise Distance Metric (2)

- Distance between traces *g* and *h*, *d*(*g*, *h*):

Manhattan length of the component-wise difference vector between ***p***(*g*) and ***p***(*h*)



$$\boldsymbol{\delta}(g,h) = \boldsymbol{p}(g) - \boldsymbol{p}(h)$$

$$d(g,h) = |\boldsymbol{\delta}(g,h)| = \sum_{i=1}^{F} |\delta_i|$$

Suspect Scores

- **Goal**: computing *suspect score* for each trace
  Largest score will be probably an anomaly

- Two cases:
  Unsupervised case: traces data only from failed execution
  Supervised case: additional data from *normal* previous run is provided

- Supervised case increases *Accuracy* of outlier detection

PURDUE UNIVERSITY

Suspect Scores — *Unsupervised*

- For each trace $h \in T$, order all traces in T according to their distance to h:

$$T_d(h) = \langle h_1, h_2, \ldots, h_{|T|} \rangle$$
$$d(h, h_i) \leq d(h, h_{i+1}),\ 1 \leq i \leq |T|$$

- The ***suspect score*** for h is the distance of h to its k^{th} nearest neighbor h_k :

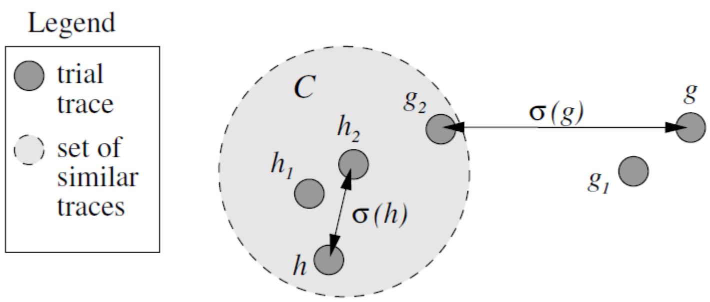
$$\sigma(h) = d(h, h_k)$$

- High suspect score ➔ Trace considered abnormal

PURDUE UNIVERSITY

Suspect Scores — *Unsupervised* (2)

- The algorithm worked well for all k larger than 3 and up to $|T|/4$.
- If $k <$ (*total number of outliers*) ➔ false negatives



$k = 2$

PURDUE UNIVERSITY

Suspect Scores — *Supervised*

- Add a set of *normal* traces N
- Arranges all traces in N in the order of their distance to h:

$$N_d(h) = \langle n_1, n_2, \ldots, n_H \rangle$$
$$d(h, n_i) < d(h, n_{i+1}),\ 1 \le i \le |T|$$

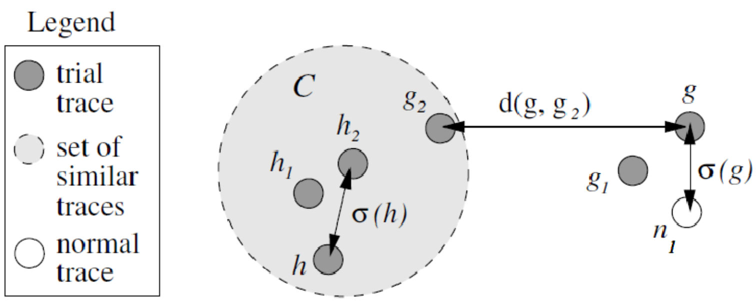
- Suspect score:

  The distance of h to either its k^{th} neighbor from T or the first neighbor from N, whichever is closer:

$$\sigma(h) = \min\{d(h, h_k),\ d(h, n_1)\}$$

PURDUE UNIVERSITY

Suspect Scores — *Supervised* (2)



$k = 2$

Problem Diagnosis Process

Collect control-flow traces until application fails

↓

Identify process that fails

↓

Root Cause Identification

Finding Cause of Anomalies

1. Last Trace Entry
   - Pinpoint the *last function* executed by the faulty host

2. Maximum component of Delta Vector
   - Component δ_i of $\delta(h, g)$ corresponds to the contribution of function f_i to the distance

$$anomFn = \operatorname*{argmax}_{1 \leq i \leq F} |\delta_i|$$

3. Anomalous Time Interval
   - Identify the first moment when the anomalous host started deviating from the norm

Experimental Test-bed

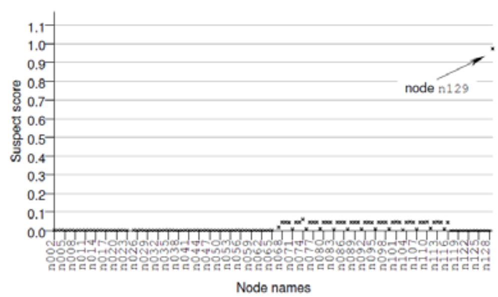
- Evaluated the techniques by locating bugs in **SCore**:
  - Large-scale cluster of workstations
  - Distributed job scheduling, checkpointing, process migration
  - 129 nodes in *Tokyo Institute of Technology*
  - C++ code base with 200,000 lines, 700 source files

Problem 1: *Network Stability*

- Symptoms:
  1. System stopped scheduling jobs
  2. Failure detected after 10 minutes and daemons were restarted
  3. Failure happened multiple times in two months
- Findings:
  - **Node 14** stopped generating trace data 500 sec earlier
  - SCore terminated by calling the **score_panic** function
  - Source code used to determined that **score_panic** was called by **freeze_sending**
  - **freeze_sending** was reported as problematic with certain NICs

PURDUE UNIVERSITY

Problem 2: *No response to requests*



- **subcast** component stopped responding to request
- Largest contribution to the node's suspect score in:

output_job_status → score_write_short → score_write → __libc_write

- Used the **Jumpshot** tool to determine application entered in a loop within last two functions

PURDUE UNIVERSITY

Summary

- Automated approach for problem determination
- Combines dynamic instrumentation and trace analysis for explaining failures
- Find the cause of problems in large-scale systems running similar tasks