

Pip: Detecting the Unexpected in Distributed Systems

Patrick Reynolds
Duke University

Charles Killian, Amin Vahdat
UC San Diego

Janet L. Wiener, Jeffrey C. Mogul, Mehul A. Shah
HP Labs, Palo Alto

UWCS OS Seminar Discussion
Andy Pavlo
07 May 2007



Problem Statement

- Distributed systems exhibit complex behaviors that can be difficult to debug:
 - Often more difficult than centralized systems.
- Parallel, inter-node activity are difficult to capture with serial, single-node tools:
 - Need something more robust than traditional profilers and debuggers.

Problem Statement

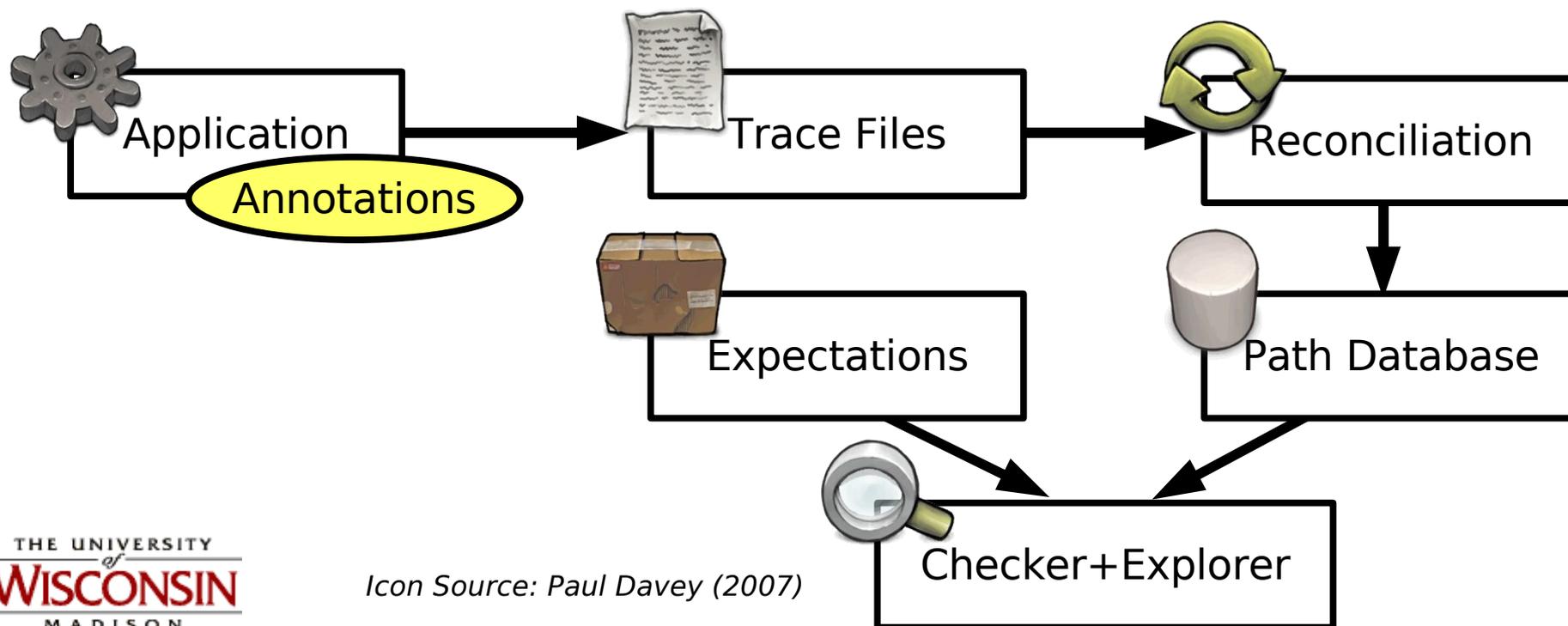
- Once behavior is captured, how do you analyze it?
- Structural bugs:
 - Application processing & communication
- Performance problems:
 - Throughput bottlenecks
 - Consumption of resources
 - Unexpected interdependencies

Pip Overview

- Suite of programs to gather, check, and display the behavior of distributed systems.
- Uses explicit path identifiers and programmer-written expectations to check program behavior.
- Pip compares actual behavior to expected behavior.

System Overview

- Annotation Library
- Declarative Expectations Language
- Trace Checker
- Behavior Explorer GUI



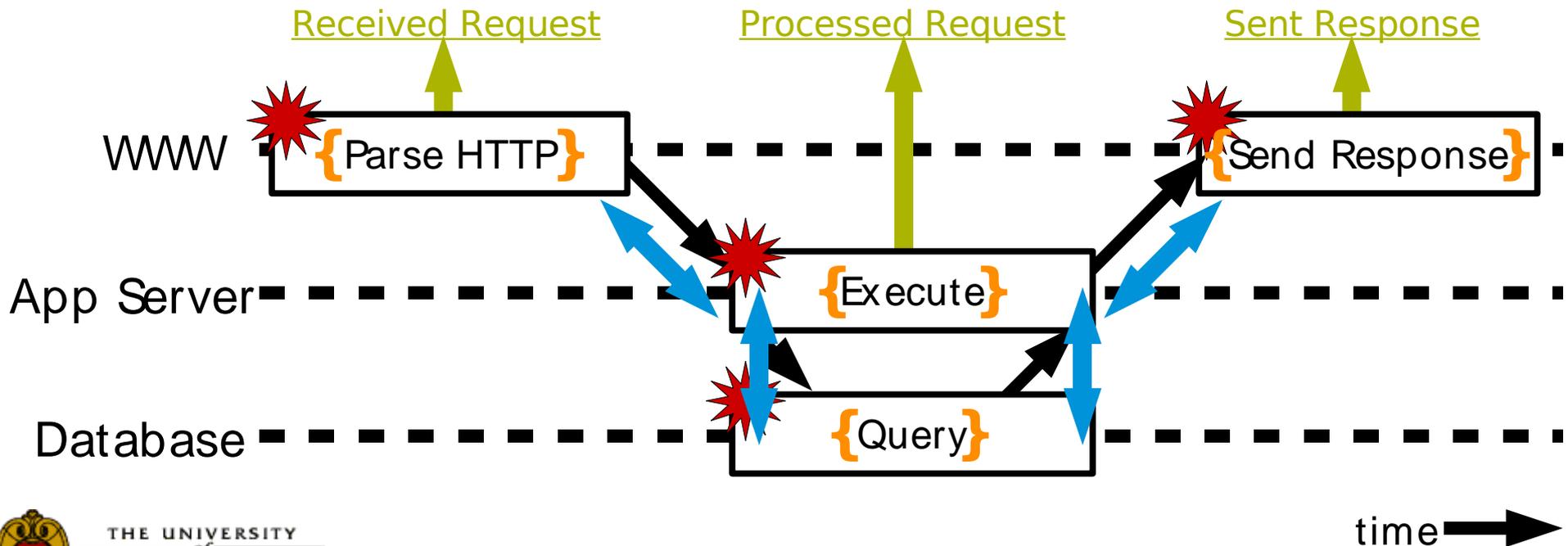
Icon Source: Paul Davey (2007)

Application Annotation

- Pip constructs an application's behavior model from generated events:
 - Manual source code annotations
 - Automatic middleware insertions
- Execution paths are based on:
 - Tasks
 - Messages
 - Notices

Application Annotation

- Set Path ID
- Start/End Task
- Send/Receive message
- Generate Notice



Expectations

- Declarative language to describe application structure, timing, and resource consumption. Expresses parallelism.
- Accommodates variation in the order and number of events for multiple paths.

```
validator CGIRequest
  task("Parse HTTP") limit(CPU_TIME, 100ms);
  notice(m/Received Request: .*\/);
  send(AppServer);
  recv(AppServer);
invalidator DatabaseError
  notice(m/Database error: .*\/);
```

Expectations

- Example: Quorum

```
validator Request
  recv(Client) limit (SIZE, {=44b});
  task("Read") {
    repeat 3 { send(Peer); }
    repeat 2 {
      recv(Peer);
      task("ReadReply");
    }
    future {
      recv(Peer);
      task("ReadReply");
    }
    send(Client);
  }
```

Expectations

- Recognizers:
 - Description of structural and performance behavior.
 - Matching
 - Matching with performance violations
 - Non-matching
- Aggregates:
 - Assertions about properties of sets of paths.

Trace Checker

- Pip generates a search tree from expectations.
- The trace checker matches results from the path database with expectations.

Behavior Explorer

- Interactive GUI displays:
 - Casual Path Structure
 - Communication Structure
 - Valid/Invalid Paths
 - Resource Usage Graphs

Behavior Explorer

Path View: fab

Tree View | Timeline View | Communication | Performance Graphs

Thread events

Thread ID: 9
Host: hpl1-021-06-eth0.hpl.hp.com
Program: srv

Time	Event
1128121548.376408	Recv(7->9, 44 bytes)
1128121548.376433	Task("fabrpc::Write")
1128121548.376538	Send(9->2, 92 bytes)
1128121548.376626	Send(9->8, 92 bytes)
1128121548.376715	Send(9->1, 92 bytes)

Task properties

Start time	1128121548.376433
End time	1128121548.379982
Real time (ms)	3.549
System time (ms)	0.000
User time (ms)	0.000
Busy %	0.000
Major faults	0
Minor faults	0
Voluntary context switches	0
Involuntary context switches	0
Starting thread	9
End thread	9

Zoom: 0.60 | Path: S{08e3c3} (8) | Host: hpl1-021-06-eth0.hpl.hp.com | Program: srv

Count	Name
1806	quorumrpc::OrderReply
1806	quorumrpc::Write2Reply
1806	quorumrpc::OrderReq
1806	quorumrpc::Write2Req
1578	quorumrpc::ReadReply
1578	quorumrpc::ReadReq

Pool	Hostname
<input checked="" type="checkbox"/> 1	hpl1-021-02-eth0.h
<input checked="" type="checkbox"/> 2	hpl1-021-03-eth0.h
<input type="checkbox"/> 3	hpl1-021-04-eth0.h
<input type="checkbox"/> 4	hpl1-021-04-eth0.h
<input type="checkbox"/> 5	hpl1-021-04-eth0.h
<input type="checkbox"/> 6	hpl1-021-04-eth0.h

ID	Path name
4	+{8608}
8	S{08e3c3}
12	n{dd9a22}
16	{27d7be05}
20){7f}{92}
24	G{1b01aa}

Filter	Recognizer	Paths	Resource vio
<input type="checkbox"/>	UnmatchedWrite	R S 0	0
<input checked="" type="checkbox"/>	Write	R F 602	0
<input checked="" type="checkbox"/>	Write3Others	V C 142	0
<input type="checkbox"/>	WriteMe1st	V C 150	0
<input type="checkbox"/>	WriteMe2nd	V C 169	0

0.0 | 111.0

Behavior Explorer

Casual Path Viewer

The screenshot displays the Behavior Explorer interface. At the top is a 'Casual Path Viewer' showing a hierarchical tree of nodes. The root node is '4'. Below it are several levels of child nodes, with some nodes highlighted in blue. The nodes contain numerical values. Below the tree is a status bar with the following information: Zoom: 0.68, Path: ransub seq 1 (101), Host: client73, Program: unit_app.

Below the main window are two detail windows:

Thread events

Thread ID: 287
Host: client73
Program: unit_app

Time	Event
1122523943.969929	Recv(4->287, 108 bytes)
1122523943.970614	Task("RanSubAggregator::deliver(ReceiveDataHandler)")
1122523943.976417	Task("RanSubAggregator::deliver::distributeMsg(ReceiveDataHandler)")
1122523943.976713	Task("recv distribute")
1122523943.976773	Notice("RanSub: dist from 0100000a seq 1, expect 1.")
1122523944.027080	Task("RanSub::distributeAggregateData(AggregateData)")
1122523944.054202	Task("ransub_test::deliverGossip(GossipDataHandler)")

Task properties

Start time	1122523943.970614
End time	1122523944.097151
Real time (ms)	126.537
System time (ms)	20.000
User time (ms)	50.000
Busy %	0.553
Major faults	1220
Minor faults	0
Voluntary context switches	0
Involuntary context switches	0
Starting thread	287
End thread	287

Executed tasks, messages, and notices

Timing & Resource Properties

Pip vs. Paradyn

- The Paradyn Configuration Language (PCL) allows programmers to describe expected characteristics of applications.
- “...PCL cannot express the casual path structure of threads, tasks, and messages in a program, nor does Paradyn reveal the program's structure”.

Using Pip in Condor

- No high-level debugging tool is currently used by Condor developers.
- Inner-working knowledge about daemon interactions is either scattered in source code documentations or with a few developers.

Discussion

- Questions?