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

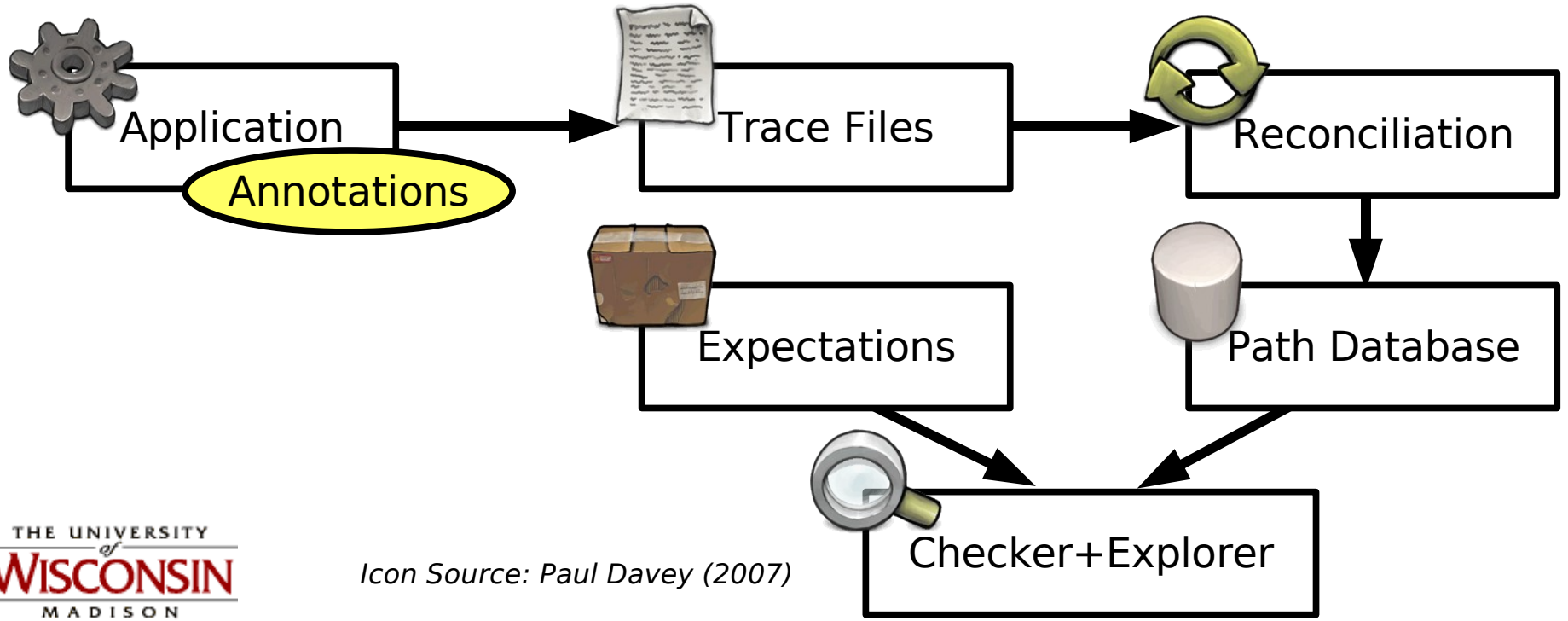
- Once behavior is captured, how do you analyze it?
- Structural bugs:
  - Application processing & communication
- Performance problems:
  - Throughput bottlenecks
  - Consumption of resources
  - Unexpected interdependencies

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



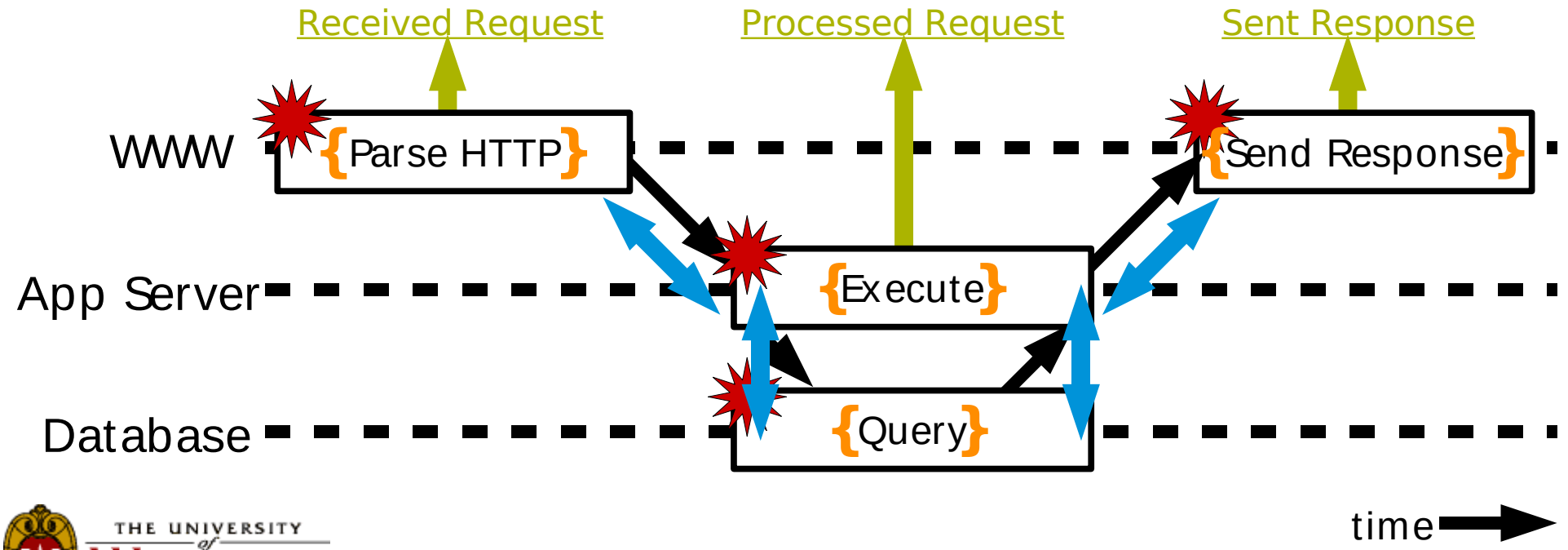
Icon Source: Paul Davey (2007)

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.

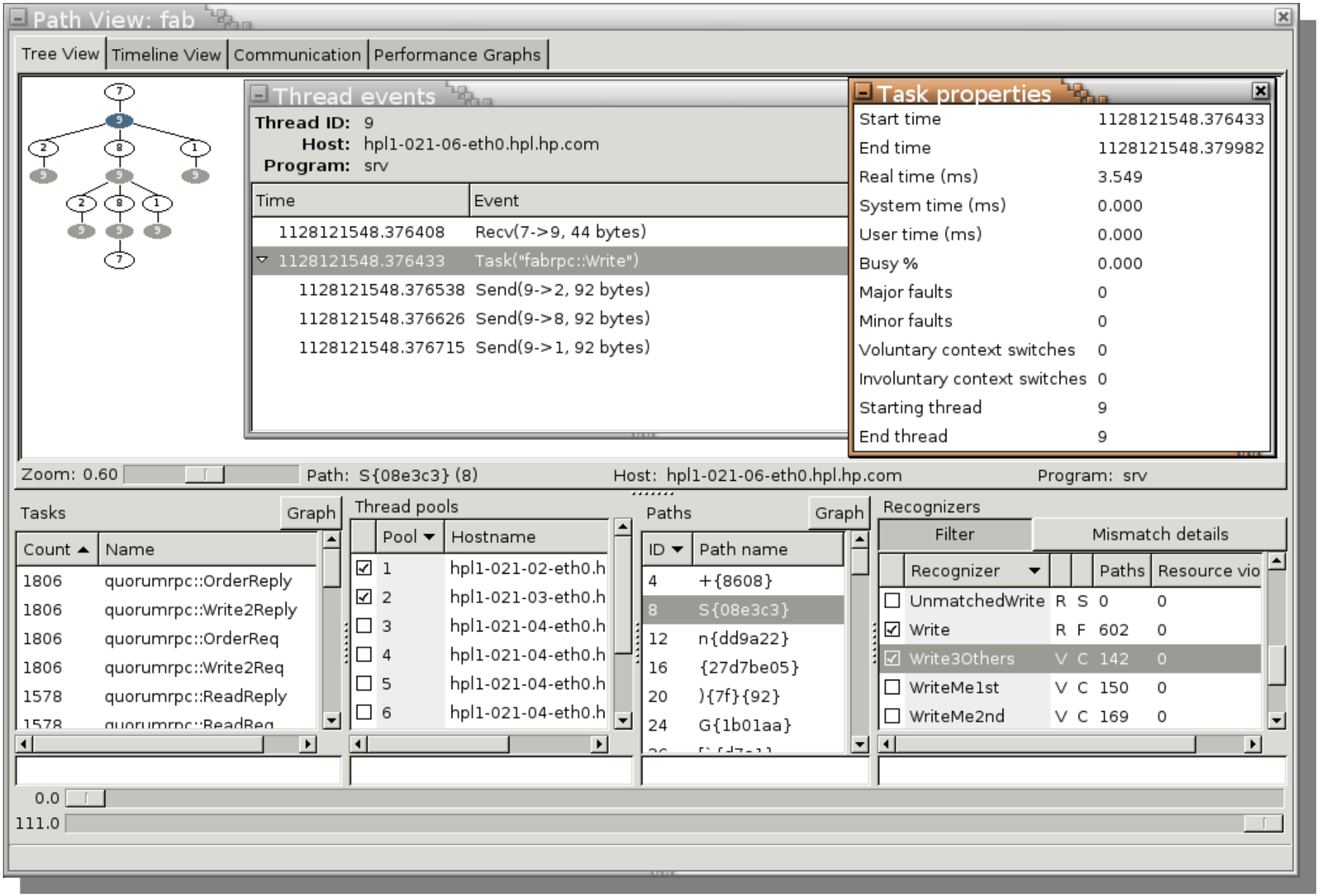
Trace Checker

- 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

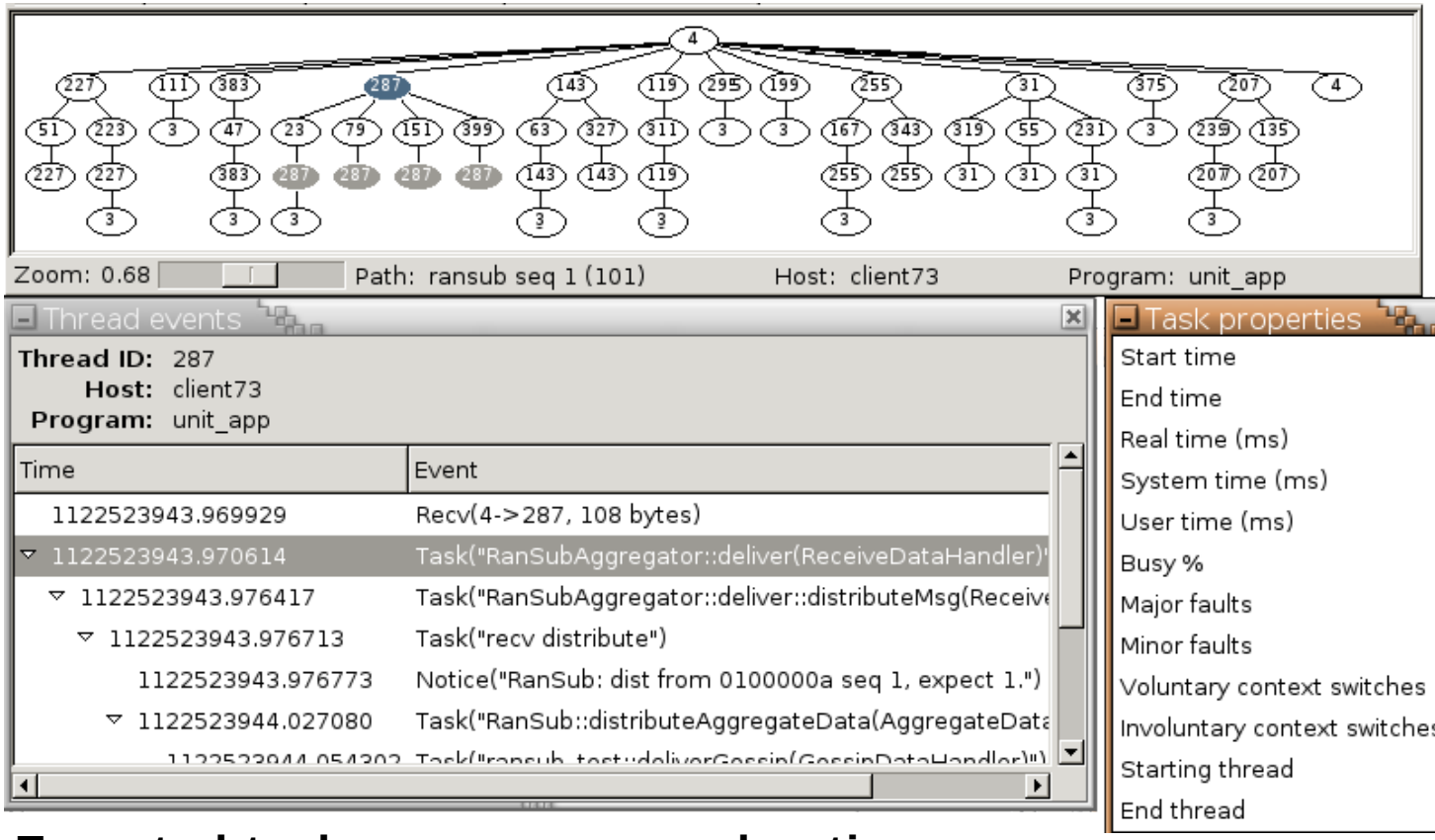
Behavior Explorer



Source: Pip web page (2007)
http://issg.cs.duke.edu/pip/

Behavior Explorer

Casual Path Viewer



Executed tasks, messages, and notices

Timing & Resource Properties

Source: Pip web page (2007)
http://issg.cs.duke.edu/pip/

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?