Towards Fingerpointing in the Emulab Dynamic Distributed System

Michael P. Kasick
Priya Narasimhan

Carnegie Mellon University

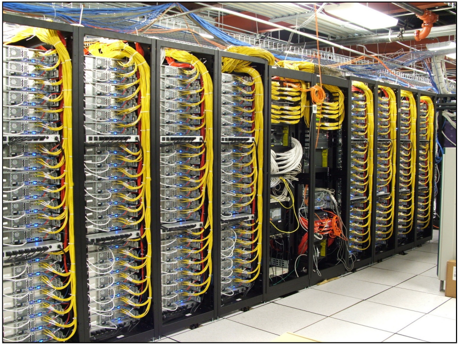
Kevin Atkinson
Jay Lepreau

University of Utah

November 5, 2006

Electrical & Computer ENGINEERING

Introduction to Emulab Classic



- University of Utah:
  Flux Research Group
- Network emulation testbed
- 1300 users
- 430 local nodes
- 740 distributed nodes
- In service for 6 years

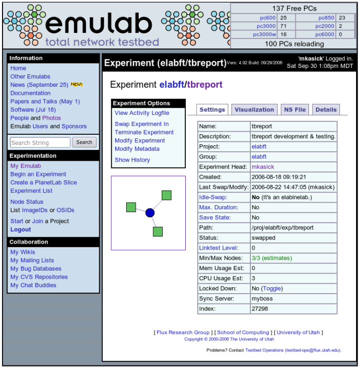
Electrical & Computer ENGINEERING

Emulab's Experiments

- Users upload an experiment configuration (NS file)
- Configuration specifies virtual node topology
- Users granted full, exclusive access to nodes
- Nodes automatically redelegated when experiments go idle

Emulab Software Infrastructure

- Off-the-shelf components
  - Database, OS, etc.
- Custom developed components
  - Web interface
  - Testbed setup & management
  - 490,000 lines of code
- Swap-* procedures
  - swap-in, swap-out, swap-modify
  - 40+ script invocations



Electrical & Computer ENGINEERING

Emulab's Difficulties

- System errors reported to operator mailing list
- Average of 82 failure emails per day (April 2006)
- Swap-* procedures are largest sources of errors
  - Each mail is 100+ lines long
  - Problem is not always obvious
  - Many underlying causes
  - Only a few errors require operator attention

Electrical & Computer ENGINEERING

Example Swap-* Failure

```
TIMESTAMP: 11:12:38:659154 assign started
assign foo-bar-5987.ptop foo-bar-5987.top
ASSIGN FAILED:
Type precheck passed.
Node mapping precheck:
Node mapping precheck succeeded
Annealing.
Fixed node: Could not map srs-101 to pc106
Trying assign on an empty testbed.
TIMESTAMP: 11:12:40:663660 ptopgen started
ptopargs -p foo -e bar -a
TIMESTAMP: 11:12:41:576498 ptopgen finished
TIMESTAMP: 11:12:41:576719 assign started
assign -n foo-bar-5987.ptop foo-bar-5987.top
Precheck succeeded.
Assign succeeded on an empty testbed.
*** /usr/testbed/libexec/assign_wrapper:
    Unretriable error. Giving up.
*** Failed (65) to map to reality.
Recovering virtual and physical state.
Doing a recovery swap-in of old state.
```

- 236 line email
- 111 lines of log
- 4 errors
- 1 root cause

Problem Summary

- Need to automate swap-* failure analysis
  - Isolate errors
  - Determine error relevance
- Can be done with post-processing analysis
- This problem solved by concurrent work
  - Emulab's new *tblog* logging mechanism

Can we do better?

Electrical & Computer ENGINEERING

Local vs. Global Analysis

- Analysis of a single swap-* failure:
  - Considers a single, local, error domain
  - Scope limits precision of fingerpointing
- Concurrent analysis of many swap-* failures:
  - Considers the entire, global, error domain
  - Error correlation increases precision of fingerpointing

Electrical & Computer ENGINEERING

The Bigger Problem

- Current error reporting does not facilitate global analysis

We propose a new *structured* error reporting mechanism that does facilitate global analysis

Outline

Electrical & Computer ENGINEERING

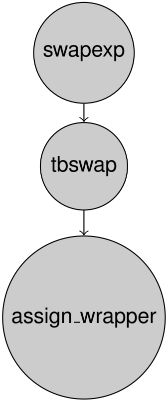
tblog Logging Mechanism

- Perl module interfaces scripts with an error-log database
- Automatically logs diagnostic messages
  - stdout, stderr, die(), warn(), etc.
- Provides an API for scripts to write messages
- Records script execution context
  - Time stamp, script invocation #, parent script #, etc.

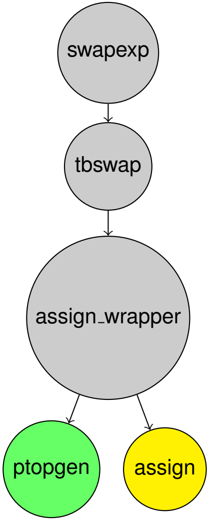
tblog Analysis

- Reconstructs script call-chain
- Ascertains most recent error at greatest depth
- Flags script and its errors as relevant

tblog Example

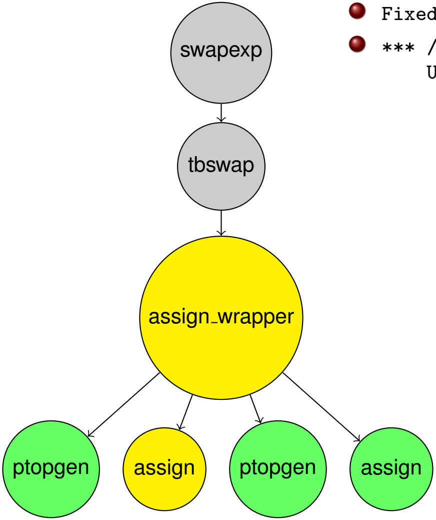


tblog Example



- `Fixed node: Could not map srs-101 to pc106`

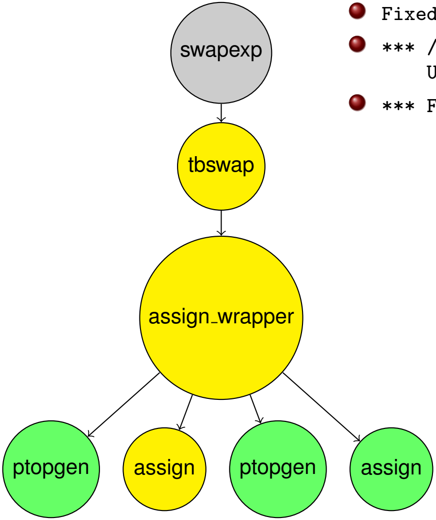
tblog Example



- `Fixed node: Could not map srs-101 to pc106`
- `*** /usr/testbed/libexec/assign_wrapper: Unretriable error. Giving up.`

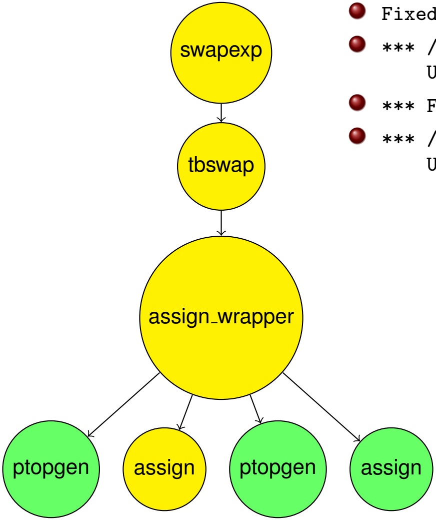
Electrical & Computer ENGINEERING

tblog Example



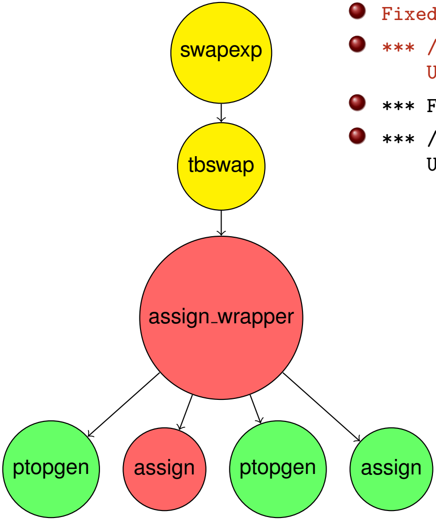
- `Fixed node: Could not map srs-101 to pc106`
- `*** /usr/testbed/libexec/assign_wrapper: Unretriable error. Giving up.`
- `*** Failed (65) to map to reality.`

tblog Example



- `Fixed node: Could not map srs-101 to pc106`
- `*** /usr/testbed/libexec/assign_wrapper: Unretriable error. Giving up.`
- `*** Failed (65) to map to reality.`
- `*** /usr/testbed/bin/swapexp: Update aborted; old state restored.`

tblog Example



- `Fixed node: Could not map srs-101 to pc106`
- `*** /usr/testbed/libexec/assign_wrapper: Unretriable error. Giving up.`
- `*** Failed (65) to map to reality.`
- `*** /usr/testbed/bin/swapexp: Update aborted; old state restored.`

Opaque Failure Messages

- Designed to be human interpretable
- Vague and lacking in context details
  - Need context for spatial correlation
  - “Unretriable error. Giving up.”
- Messages are cumbersome to parse
- Different messages may describe the same error
  - “Invalid OS FOO in project bar!”
  - “[tb-set-node-os] Invalid osid FOO”

Electrical & Computer ENGINEERING

Absence of Error Context

- *tblog* only captures a general context
  - time stamp, script name, etc.
- No provision for capturing error-specific context
  - nodes, OS images, configuration, etc.
- Reporting must preserve the error-specific context
  - Required for error correlation
  - Facilitates global analysis

Outline

Electrical & Computer ENGINEERING

Discrete Error Types

- Identify distinct errors
- Defined by a specification for each error type
  - named error type, definition, associated specific context
- Node-boot failure example:
  - Error type: node_boot_failed
  - Definition: Node failed to boot twice
  - Context: node, type, osid

Electrical & Computer ENGINEERING

Error Context & Propagation

- Context distinguishes between errors of the same type
  - Node boot failures across different nodes
  - Node boot failures with different OSes
- Propagation centers focus on relevant errors
  - Nested scripts should propagate the primary error
  - Otherwise parent scripts generate “me-too” errors
  - Secondary (“me-too”) errors add noise
  - Achievable with exceptions (RPC, middleware)

Research Phase

- Used *tblog* to identify a set of target errors
  - Goal was not to obtain 100% coverage
  - System functionality is always expanding
  - Small portion of possible errors actually observed
- Drafted error specifications and error types
  - Required significant knowledge of errors and meaning
  - Eliminated error ambiguities
  - Identified relevant error specific context

Development Phase

- Developed a prototype Perl reporting module
  - Structured error reporting function
  - Error parsers for C++ & TCL language components
- Added reporting hooks for the target errors
  - Problem: Emulab provides no error propagation
  - Nested scripts return success or failure only
  - Fix: severity-level assignment
  - Alternative: *tblog* post-processing analysis

Testing & Deployment Phases

- Tested prototype in *elabinelab*
- Integrated prototype into *tblog* framework
  - New local analysis engine: *tbreport*
- Deployed on the production Emulab testbed
- 750 lines of added or changed code

Initial Results

- Data collected August 16-24th, 2006
- 681 swap-* sessions started
  - 108 (17.3%) reported at least one error
- 283 total fatal errors reported
  - Many errors repeated for each node in a session
  - 118 unique instances of errors in a given session

Electrical & Computer ENGINEERING

Error Statistics

Occurrences		Error Type
31	26.3%	assign_violation/feasible (resource shortage)
24	20.3%	assign_type_precheck/feasible (node shortage)
22	18.6%	node_boot_failed
10	8.5%	ns_parse_failed (bad experiment configuration)
...	...	...

Normalized errors (unique in a session) grouped by error type.

Node Shortage Failures

- Second most common error (20.3%)
- Insufficient free nodes for experiment swap-in
- Current node availability is listed on website

59 Free PCs			
pc600	6	pc850	4
pc3000	25	pc2000	8
pc3000w	16	pc6000	0
1 PCs reloading			

- Illustrates user demand
  - 48% due to lack of pc3000

Electrical & Computer ENGINEERING

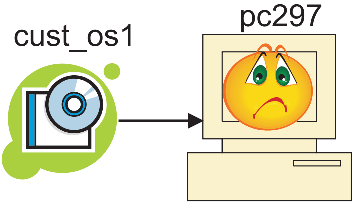
Other Resource Shortage Failures

- Most common error (26.3%)
- Sufficient free nodes to swap-in, but:
  - Attempted assignment violated mapping constraints
  - Often due to oversubscribed switch bandwidth
- Assignment algorithm is non-deterministic
  - User cannot predict when these errors might occur
  - Later attempts may succeed w/o topology change
- Frequent resubmissions lead to further errors

Node-Boot Failures

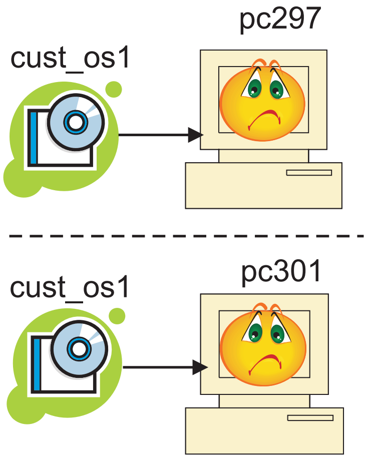
- Third most common error (18.6%)
- Node status daemon
  - Reports boot success
  - Timeout results in error
- Many underlying causes
  - Faulty hardware, broken user contributed OS, etc.
- Motivating scenario for our future research

Node-Boot Failure Example (I)



- Single node, one session
- Unknown culprit

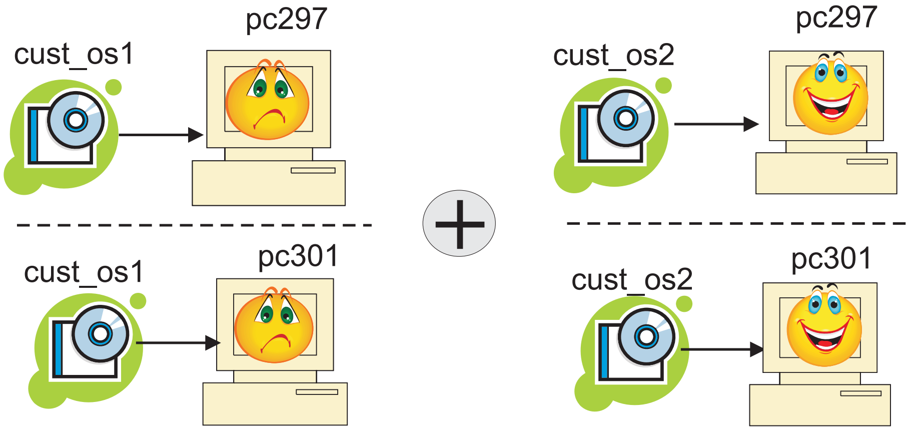
Node-Boot Failure Example (II)



- Two nodes, two sessions, same OS
- Suggests bad OS

Electrical & Computer ENGINEERING

Node-Boot Failure Example (III)



- Same two nodes, four sessions, different OS
- Strongly suggests bad OS

Node-Boot Failures: What's Next?

- Cannot diagnose root cause from a single trace
- Operator dilemma:
  - Assume node is faulty and quarantine?
  - Assume OS is faulty and leave node as is?
- Motivates global fingerpointing (future work)
  - Correlation of multiple error instances
  - Reliably fingerpoints the culprit

Electrical & Computer ENGINEERING

Summary

- Manual diagnosis of system errors is costly
- *tblog*-style analysis aids in message filtering
- Opaque failure messages limits error usefulness
- Structured error reports enable global analysis
- Global analysis fingerpoints errors with fine granularity
- Future work:
  - Develop a global analysis engine for Emulab
  - Start by targeting the identified node-boot failure scenario
  - Target other real-world systems for error analysis

Electrical & Computer ENGINEERING

Further Reading

Michael P. Kasick, Priya Narasimhan, Kevin Atkinson, and Jay Lepreau.
Towards fingerpointing in the Emulab dynamic distributed system.
In *Proceedings of the 3rd USENIX Workshop on Real, Large Distributed Systems (WORLDS '06)*, Seattle, WA, November 2006.

Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar.
An integrated experimental environment for distributed systems and networks.
In *Proceedings of the Fifth Symposium on Operating System Design and Implementation (OSDI '02)*, pages 255–270, Boston, MA, December 2002.