Experiences with X-Trace: an end-to-end, datapath tracing framework

Sun Microsystems Laboratories
January 30, 2008

George Porter, Rodrigo Fonseca, Matei Zaharia, Andy Konwinski, Randy H. Katz, Scott Shenker, Ion Stoica

X-Trace

• Framework for capturing causality of events in a distributed system
  – Coherent logging: events are placed in a causal graph
    • Capture causality, concurrency
    • Across layers, applications, administrative boundaries

• Audience
  – Developers: debug complex distributed applications
  – Operators: pinpoint causes of failures
  – Users: report on anomalous executions
A user gets a stale page. What went wrong?

Combinatorial explosion of possible paths through the app.

Well Known Problem

- Disconnected logs in different components
- Multiple problems with the same symptoms
- Execution paths are ephemeral

* Troubleshooting distributed systems is **hard**
Coral CDN

- Open, distributed content distribution network
  - Distributed cache, uses a self-organizing, locality-aware DHT
  - Usage is simple: append .nyud.net to domain in url
  - +25M requests/day, runs on > 250 Planetlab nodes
  - Built with libasync

A Coral Request

- Interesting case for tracing
  - Involves recursive DNS, HTTP, RPC (and ICMP)
  - We trace DNS processing, HTTP (incl recursive), and RPC
Adding X-Trace

- Capture events within application
  - Logging API
  - Capture abstraction
  - Capture parallelism

Adding X-Trace

- Capture events on different layers
  - e.g. HTTP and RPC
Adding X-Trace

- Correlate events
  - Across different machines
  - Across different layers

X-Trace Mechanisms

- Each Task gets a unique TaskId
- Each Event within a task get a unique EventId
- When logging, each event must record “edge”:
  
  previous EventId > new EventId

- <TaskId, last EventId> propagated with execution
  - Within runtime environment (X-Trace libraries)
  - Through API calls (augmented APIs)
  - In protocol messages (as header metadata)
**Trace collection and storage**

- Trace data buffered and distributed across instrumented hosts
- Collection process
  - Orthogonal
  - Minimize collection overhead via buffering and compression

**Explicit Path Tracing**

- **Advantages**
  - Deterministic causality and concurrency
  - *Handle* on specific executions (name the needles)
  - Does not depend on time synchronization
  - Correlated logging
    - *Meaningful* sampling (random, biased, triggered...)
- **Disadvantages**
  - Modify applications and protocols (some)
Talk Roadmap

- X-Trace motivation and mechanism
- Use cases:
  1. Wide-area: Coral Content Distribution Network
  2. Enterprise: 802.1X network authentication
  3. Datacenter: Hadoop Map/Reduce
- Future work within the RAD Lab
  - Debugging and performance
  - Clustering and analysis of relationships between traces
  - Applying tracing to energy conservation and datacenter management

Use Cases

1. Wide-area: Coral CDN

2. Enterprise: 802.1X Network Authentication

3. Datacenter: Hadoop Map/Reduce
Coral Deployment

- Running on production Coral network since Christmas
- 253 machines
- Sampling: tracing 0.1% of requests

Initial Findings

- Found at least 5 bugs :-)
  - “Wrong timeout value for proxy”
  - “Some paths when server fetch fail may not kill client connection”
  - “Does Coral lookup even when likely to be over cache size”
  - “Forwarding internal state to client”
  - “Revalidation always goes to origin servers, not to peers”

- Some timing issues
  - Very slow HTTP responses, investigating cause
Time of HTTP Responses

- 1. Client timeout, large object, slow node, 1 block
- 2. Very slow link to client
- 3. Very slow Coral node
- 4. Failure to connect to origin server, specific 189s timeout

Can look at graph for each specific point, e.g.

Use Cases

1. Wide-area: Coral CDN
2. Enterprise: 802.1X Network Authentication
3. Datacenter: Hadoop Map/Reduce
Enterprise: IEEE 802.1X

- Controls user access to network resources
  - Wireless access points
  - VPN endpoints
  - Wired ports
- User-specific admission criteria
- Audit for compliance purposes

Complex protocol
- Distributed for scalability and reliability: no central point
- Multi-protocol
- Spans administrative domains
- Multi-vendor

802.1X Overview

1. Client sends credentials to authenticator
2. Authenticator forwards credentials to authentication server with RADIUS
3. Authentication server queries identity store with LDAP
4. Identity store processes query
5. Identity store responds with success or failure using LDAP
6. Authentication server makes decision; sends RADIUS response to Authenticator
7. Authentication server receives response
8. Access is granted or denied
802.1X and X-Trace: how to read a trace of a successful request

Approach

• Collect application traces with X-Trace
• Determine when a fault is occurring
• Localize the fault
• **Determine the root cause, if possible**
• Report problem and root cause (if known) to network operator
**Root cause determination**

- Why these tests?
  - Occur in customer deployments
  - Based on conversation with support technicians

**Root test 1: insufficient timeout setting**

- Timeouts spread throughout system; set by different administrators, vendors
- Detection: Authenticator times out at $T_1$, Radius server issues report at $T_2 > T_1$
- Radius timeout at $T_1 = 2.034s$
- LDAP responds at $T_2 = 3.449s > T_1$
Root test 2: udp packet loss (reverse path)

- Radius protocol based on udp: packet loss manifests as a dropped message
- Detection: evidence of forward progress coupled with timeout and retry

Inferring network failures with application traces

- Methodology for inferring network and network service failures from application traces
- Beneficial to 802.1X vendor
  - Network appliances installed in foreign network
  - Lack of direct network visibility

1. Administrative division of responsibility
   - App developer separate from network operator
2. Failures only detectable from application
3. Virtualized datacenter
   - Network implementation hidden by design
Use Cases

1. Wide-area: Coral CDN
2. Enterprise: 802.1X Network Authentication
3. Datacenter: Hadoop Map/Reduce

Apache Hadoop

- Open-source implementation of Google File System + Map/Reduce
  - Map/Reduce - parallel computation
  - HDFS - distributed block store
- Architecture
  - Single namenode that stores file system metadata
  - Cluster of datanodes that store replicated blocks
Approach

- Instrument Hadoop using X-Trace
- Analyze traces in web-based UI
- Detect problems using machine learning
  1. Identify bugs in new versions of software from behavior anomalies
  2. Identify nodes with faulty hardware from performance anomalies

Hadoop-specific trace display

- Web-based
  - Built on top of X-Trace backend
- Provides
  - Performance statistics for RPC and DFS ops
  - Graphs of utilization, performance versus various factors, etc
  - Critical path analysis: breakdown of slowest map and reduce tasks
1. Detecting suboptimal default configuration with behavior anomalies

**Problem:** One single Reduce task, which actually fails several times at the beginning

After optimizing configuration

Active tasks vs. time with improved configuration

(50 reduce tasks instead of one)
2. Detecting fault underlying servers with performance anomalies

Diagnosed to be failing hard drive

Functional Contingency Table

<table>
<thead>
<tr>
<th>1st func/</th>
<th>following func</th>
<th>exists</th>
<th>OP_WRITE_BLOCK</th>
<th>mkdirs</th>
<th>open</th>
<th>getFileInfo</th>
<th>getBlockLocations</th>
<th>rename</th>
</tr>
</thead>
<tbody>
<tr>
<td>exists</td>
<td></td>
<td>798</td>
<td>1168</td>
<td>220</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>create</td>
<td></td>
<td>565</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>complete</td>
<td></td>
<td>512</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mkdirs</td>
<td></td>
<td>52</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getFileInfo</td>
<td></td>
<td>171</td>
<td>366</td>
<td>481</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delete</td>
<td></td>
<td>53</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OP_READ_BLOCK</td>
<td></td>
<td>1</td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getBlockLocations</td>
<td></td>
<td>325</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>getListing</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Represents pairs of adjacent events in the trace
  - Can span multiple datanodes as well as datanode/namenode interactions
- Simple technique, but can find bugs
  - Tested with old versions of the code, checked for bugs found and patched in later versions
Hadoop Summary

- Initial results promising
  - Check out old code, run the program, see if we caught the bug that was fixed in a later version
  - Both Hadoop bugs as well as underlying infrastructure failures
- Ongoing work with large Hadoop datasets at Facebook

Talk Roadmap

- X-Trace motivation and mechanism
- Use cases:
  1. Wide-area: Coral Content Distribution Network
  2. Enterprise: 802.1X network authentication
  3. Datacenter: Hadoop Map/Reduce
- Future work within the RAD Lab
  - Debugging and performance
  - Clustering and analysis of relationships between traces
  - Applying tracing to energy conservation and datacenter management
D^2aiquiri

• Trace a subset of the tasks
  – Supported by X-Trace’s coherent sampling
• Use probabilistic modeling to infer what most likely happened for the traces you didn’t measure
• Benefits:
  – Track service and wait time of each component with less trace data than before
  – Or, given a set of trace data, infer the result of a much more invasive trace
• Probabilistic queuing model of distributed system
  – Random variables: Arrival and departure of each component
• Sampler of arrivals given observations
• From samples, compute for each queue
  – Expected waiting time: time due to load
  – Expected service time: intrinsic processing

\[ T = W + S \]

D3: Declarative Distributed Debugging

• Process logs in a distributed fashion
  – Send queries to nodes
  – Process logs locally at each node
  – Concise results back to front node
• Specify queries declaratively
  – Uses P2 distributed database
  – Variant of Datalog
  – Execution tracing, performance, distributed assertions
• Great reduction in traffic
  – Hadoop example: 4.5Gb logs vs 65KB for performance query
Conclusions / Status

• Open framework for capturing:
  – Causality
  – Abstraction
  – Layering
  – Parallelism

• Result:
  – Datapath trace spanning machines, processes, and software components
  – Can “name”, store, and analyze offline

• Libraries, software, and documentation available under BSD license

• We welcome integration opportunities -- contact us!

• Thank you

www.x-trace.net

Backup
Software Bug Detection

• Generate *contingency tables* containing counts of X-Trace graph features (e.g. adjacent events)
• Compare test run against a set of “good” runs
• Scenarios:
  – Simulated software bugs: random `System.exit()`
  – Real bug in a previous version of Hadoop
• Statistical tests
  – Chi-squared (three variations)
  – Naïve Bayes

D2aiquiri
Approach

Split up response time at each component:

M/M/1 queue

For each task $k$:

- Arrival time $a_k$
- Departure time $d_k = a_k + w_k + s_k$
- Service time $s_k \sim \text{Exp}(\mu)$
- Waiting time $w_k = \max\{0, d_{k-1} - a_k\}$

Probabilistic Modeling

Now we have a probability distribution over arrivals, departures, service times, and waiting times.
Arrival and departure times can be instrumented. 

**Question**: Can we reconstruct missing arrivals?  
**Hypothesis**: Use conditional distribution
Instrumentation Challenges

• Application-level multiplexing/scheduling
  – Mixed tasks that had requests placed in a queue
  – Mixed tasks in a multiplexing async DNS resolver
  – Fix: instrumented the resolver and the queue to keep track of the right metadata

• Unnecessary Causality
  – Some tasks trigger periodic events (e.g. refresh)
  – Fix: we can explicitly stop tracing these

Open Question

• Tradeoff between intrusiveness and expressive power
  – Black-box: inference-based tools
  – Gray-box: automatic instrumentation of middleware
  – White-box: application-level semantics, causality, concurrency

• Question: what classes of problems need the extra power?
Ongoing work

- Refining logging API
- Integration with more software
- Defining analysis and queries we want
- Distributed logging infrastructure
- Graph analysis tools and APIs
  - *E.g.*, *is this a typical graph?*
- Distributed querying
  
- Integration with telemetry data
- Automating tracing as much as possible

Oasis Anycast Service

- General anycast service:
  
- “Find me a server of service X close to me”
  - Shared by many services
  - Works via DNS resolution of HTTP redirect
  - Built with libasync, runs on Planetlab

- Instrumentation
  - Each DNS and HTTP request generates new Task
  - Some more application level tracing
  - Automatic tracing through RPCs (libarpc)
Oasis DNS Request

• Typical DNS resolution with two ‘threads’

Sample X-Trace report

X-Trace Report ver 1.0
Host: planetlab3.millennium.berkeley.edu
RPC: oasclnt_program_1-1:OASCLNTPROC_RECS
Edge: 0187A463, next
Agent: arpc clnt
Timestamp: 1191545172.213 [event]
Label: send call
X-Trace: 00BFA5373F31575435

Very little trouble...

• Oasis Anycast service
  – Instrumented 20 nodes on Planetlab for 1 week
  – Performance of Oasis was great (99.9% < 10 sec)
...but some trouble

- Tail was due to some specific timeouts in the software, and some laggard machines
- Did find two “interesting” behaviors
  - Unnecessary lookups: A to B, C, B, C
  - Async PUT out of control
    - After a request, schedule a timer to store in DHT (PUT)
    - Normally a few PUTs (for reliability)
    - In one case, 269!

Oasis PUT Bug
Anomaly Detection

- Transition graph, aggregated over many tasks
  - Initial attempts at anomaly detection

Time-sync correction

- X-Trace “edges” are timestamped and causally related => can emulate NTP
X-Trace Logging API

• X-Trace Context
  – per-thread, globally accessible X-Trace Metadata (Task Id, last event Id)

• X-Trace Event
  – log event, advance X-Trace Context

API (continued)

• Behind the scenes:
High level view

• Receive message
  – extract or create X-Trace metadata
  – set X-Trace context

• Any subsequent point
  – create X-Trace event report
  – reports will be causally related

• Any message you send
  – add X-Trace metadata to follow through