Profiling and diagnosing large-scale decentralized systems

David Oppenheimer

ROC Retreat
Thursday, June 5, 2003
Why focus on P2P systems?

- There are a few real ones
  - file trading, backup, IM

- Look a *lot* like other decentralized wide-area systems
  - Grid, sensor networks, mobile ad-hoc networks, ...

- Look a *little* like all wide-area systems
  - geog. dist. Internet services, content distribution, networks, federated web services, *@home, DNS, BGP, ...

- Good platform for prototyping services that will eventually be deployed on a large cluster

- P2P principles seeping into other types of large systems (corporate networks, clusters, ...) 
  - self-configuration/healing/optimization
  - decentralized control

- Large variability (in configurations, software versions, ...) justifies a rich fault model
Why focus on P2P systems? (cont.)

- This is NOT about the DHT abstraction
- DHT research code just happens to be the best platform for doing wide-area networked systems research right now
What’s the problem?

• Existing data collection/query and fault injection techniques not sufficiently robust and scalable for very large systems in constant flux
  ⇒ goal: enable cross-layer decentralized system profiling
    - decentralized data collection
    - decentralized querying
    - online data collection, aggregation, analysis

• Detecting and diagnosing problems is hard
  ⇒ goal: use benchmark data collection/analysis infrastructure to detect/diagnose problems (< TTD/TTR)
  ⇒ observation: abnormal component metrics (may) indicate an application or infrastructure problem
    - distinguishing normal from abnormal per-component and per-request statistics
Benchmark metrics

- **Visible at user ↔ application interface**
  - latency, throughput, precision, recall
- **Visible at application ↔ routing layer interface**
  - latency and throughput to \{find object's owner, route msg to owner, read/write object\}, latency to join/depart net
- **Cracking open the black box**
  - per-component and per-request consumption of CPU, memory, net resources; # of requests component handles; degree of load balance; # of replicas of data item
- **Recovery time, degradation during recovery**
  - recovery time broken into TT\{detect, diagnose, repair\}
- **Philosophy: collect fine-grained events, aggregate later as needed**

<table>
<thead>
<tr>
<th></th>
<th>per-component</th>
<th>across all components</th>
</tr>
</thead>
<tbody>
<tr>
<td>per-request</td>
<td>collect</td>
<td>aggregate</td>
</tr>
<tr>
<td>across all requests</td>
<td>aggregate</td>
<td>aggregate</td>
</tr>
</tbody>
</table>
Querying the data: simple example

(SQL used for illustration purposes only)

app-level request sends

<table>
<thead>
<tr>
<th>nodeID</th>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1</td>
<td>5:0.18</td>
</tr>
<tr>
<td>x1</td>
<td>2</td>
<td>10:0.01</td>
</tr>
<tr>
<td>x1</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

app-level response receives

<table>
<thead>
<tr>
<th>nodeID</th>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>x1</td>
<td>1</td>
<td>5:0.28</td>
</tr>
<tr>
<td>x1</td>
<td>2</td>
<td>10:0.91</td>
</tr>
<tr>
<td>x1</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

```
SELECT avg(KR.time-KS.time)
FROM KR, KS
WHERE KR.id = KS.id AND
     nodeID = x1
```

application

DHT storage

routing

node x1

node x2

node x3

node x4
Schema motivation

• Popular programming model is stateless stages/components connected by message queues
  - “event-driven” (e.g., SEDA), “component-based,” “async”

• Idea: make the monitoring system match
  - record activity one component does for one request
    » starting event, ending event

• Moves work from collection to query time
  - this is good: slower queries are OK if means monitoring won’t degrade the application
Monitoring “schema”
(tuple per send/rcv event)

<table>
<thead>
<tr>
<th>data item</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>operation type (send/receive)</td>
<td>1</td>
</tr>
<tr>
<td>my node id</td>
<td>4</td>
</tr>
<tr>
<td>my component type</td>
<td>4</td>
</tr>
<tr>
<td>my component id</td>
<td>8</td>
</tr>
<tr>
<td>global request id</td>
<td>16</td>
</tr>
<tr>
<td>component sequence #</td>
<td>4</td>
</tr>
<tr>
<td>request type</td>
<td>4</td>
</tr>
<tr>
<td>time msg sent/received</td>
<td>8</td>
</tr>
<tr>
<td>msg size</td>
<td>8</td>
</tr>
<tr>
<td>arguments</td>
<td>&gt;4</td>
</tr>
<tr>
<td>return value</td>
<td>4</td>
</tr>
<tr>
<td>message contents</td>
<td>256</td>
</tr>
</tbody>
</table>

(send table only)

<table>
<thead>
<tr>
<th>data item</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>peer node id</td>
<td>4</td>
</tr>
<tr>
<td>peer component id</td>
<td>4</td>
</tr>
<tr>
<td>memory consumed this msg</td>
<td>4</td>
</tr>
<tr>
<td>CPU consumed this msg</td>
<td>4</td>
</tr>
<tr>
<td>disk consumed this msg</td>
<td>4</td>
</tr>
<tr>
<td>net consumed this msg</td>
<td>4</td>
</tr>
</tbody>
</table>

What is data rate? [10k-node system, 5k req/sec]

» ~28 msgs/req * 5000 req/sec = 140,000 tuples/sec (=>14tps/node)
» ~50B/tuple * 140,000 tuples/sec = ~53 Mb/sec (=>5.5 Kbps/node)
Decentralized metric collection

"I sent req 4 at 10 AM"
Querying the data

• **Version 0 (currently implemented)**
  - log events to local file
  - fetch everything to querying node for analysis (scp)

• **Version 1 (use overlay, request data items)**
  - log events to local store (file, db4, ...)
  - querying node requests **data items** for local processing using “sensor” interface
  - **key** could be query ID, component ID, both, other...
  - overlay buys you self-configuration, fault-tolerance, network locality, caching
  - two modes
    - pull based (periodically poll)
    - push based (querying node registers continuously-running proxy on queried node(s))
Querying the data, cont.

- **Version 2 (use overlay, request predicate results)**
  - log events to local store (file, db4, ...)
  - querying node requests **predicate results** from end-nodes
    » queried node can filter/sample, aggregate, ..., before send results
    » allows in-network filtering, aggregation/sampling, trigger
    » can use to turn on/off collecting specific metrics, nodes, or components
    » SQL translation: push SELECT and WHERE clauses

- two modes
  » pull based
  » push based

- **Goal is to exploit domain-specific knowledge**
What's the problem?

• Existing data collection/query and fault injection techniques not sufficiently robust and scalable for very large systems in constant flux

  ⇒ **goal**: enable cross-layer decentralized system profiling
    - decentralized data collection
    - decentralized querying
    - online data collection, aggregation, analysis

• **Detecting and diagnosing problems is hard**

  ⇒ **goal**: use benchmark data collection/analysis infrastructure to detect/diagnose problems (< TTD/TTR)

  ⇒ **observation**: abnormal component metrics (may) indicate an application or infrastructure problem

  - distinguishing normal from abnormal per-component and per-request statistics
What the operator/developer wants to know

1. Is there a problem?
   - s/w correctness bug, performance bug, recovery bug, hardware failure, overload, configuration problem, ...

2. If so, what is the cause of the problem?

Currently: human involved in both
Future: automate, and help human with, both
Vision: automatic fault detection

- Continuously-running queries that generate alert when exceptional conditions are met
  - example: \( \text{avg application response time during last minute} > 1.1 \times \text{avg response time during last 10 minutes} \)

```
[now = 11:0.0]

SELECT "alert" AS result WHERE
  (SELECT avg(KR.time-KS.time) FROM KR[Range 1 Minute], KS WHERE KR.id=KS.id) > 1.1 *
  (SELECT avg(KR.time-KS.time) FROM KR[Range 10 Minute], KS WHERE KR.id=KS.id)

0:0.90 > 1.1 \times 0:0.50 ? ALERT!
```

<table>
<thead>
<tr>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5:0.18</td>
</tr>
<tr>
<td>2</td>
<td>10:0.01</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5:0.28</td>
</tr>
<tr>
<td>2</td>
<td>10:0.91</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Status: essentially implemented (for a few metrics)

- Built on top of event logging + data collection infrastructure used for the benchmarks
- Not yet implemented: threshholding
  - currently just collects and graphs the data
  - human generates alert using eyeballs and brain
Vision: automatic diagnosis (1)

- Find request that experienced highest latency during past minute

[now = 11:0.0]

<table>
<thead>
<tr>
<th>KS</th>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5:0.18</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10:0.01</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>KR</th>
<th>req id</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5:0.28</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10:0.91</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

```
SELECT KR.time-KS.time, KR.id as theid
FROM KR[Range 1 Minute], KS[Range 1 Minute]
WHERE KR.id=KS.id AND KR.time-KS.time = (SELECT max(KR.time-KS.time)
FROM KR[Range 1 Minute], KS[Range 1 Minute]
WHERE KR.id = KS.id)
```

0:0.90, theid = 2

[we will investigate this request on the next slide]
Vision: automatic diagnosis (2)

- How long did it take that message to get from hop to hop in the overlay?

IS, IR tables: decentralized routing layer sends/receives

<table>
<thead>
<tr>
<th>req id</th>
<th>time</th>
<th>me</th>
<th>nexthop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10:00.05</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>...</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>A</td>
<td>...</td>
</tr>
</tbody>
</table>

IS (node A)

<table>
<thead>
<tr>
<th>req id</th>
<th>time</th>
<th>me</th>
<th>exthop</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>...</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>11</td>
<td>10:08.5</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>23</td>
<td>...</td>
<td>B</td>
<td>E</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>B</td>
<td>...</td>
</tr>
</tbody>
</table>

IR (node A)

<table>
<thead>
<tr>
<th>req id</th>
<th>time</th>
<th>me</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>...</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>...</td>
<td>A</td>
</tr>
<tr>
<td>23</td>
<td>...</td>
<td>B</td>
</tr>
</tbody>
</table>

SELECT IR.time - IS.time as latency, IS.me as sender, IR.me as receiver
WHERE IS.nexthop = IR.me AND IS.id = 2 AND IR.id = 2

- latency = ..., sender = ..., receiver = A
- latency = 0.80, sender = A, receiver = B
- latency = ..., sender = B, receiver = ...
Status: manual “overlay traceroute”

- Simple tool to answer previous question
  - “How long did it take that message to get from hop to hop in the overlay?”
- Built on top of event logging+data collection infrastructure used for the benchmarks
- Only one metric: overlay hop-to-hop latency
- Synchronizes clocks (currently out-of-band)
- Operates passively
- No fault injection experiments yet; coming soon

<table>
<thead>
<tr>
<th>optype</th>
<th>reporting_node</th>
<th>request_id</th>
<th>report_time</th>
<th>diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>inject</td>
<td>169.229.50.219</td>
<td>3@169.229.50.219</td>
<td>1054576732997161</td>
<td></td>
</tr>
<tr>
<td>forward</td>
<td>169.229.50.223</td>
<td>3@169.229.50.219</td>
<td>1054576732998725</td>
<td>1564</td>
</tr>
<tr>
<td>forward</td>
<td>169.229.50.213</td>
<td>3@169.229.50.219</td>
<td>1054576733008831</td>
<td>10106</td>
</tr>
<tr>
<td>forward</td>
<td>169.229.50.226</td>
<td>3@169.229.50.219</td>
<td>1054576733021493</td>
<td>12662</td>
</tr>
<tr>
<td>deliver</td>
<td>169.229.50.214</td>
<td>3@169.229.50.219</td>
<td>1054576733023786</td>
<td>2293</td>
</tr>
</tbody>
</table>
Building and using behavioral profiles

- Benchmarks measure behavioral profile for fixed w/load
- **Goal is to automate problem detection/diagnosis**
  - too much data for a human to do it manually
- **Version 0 (human builds and applies model)**
  - human detects and diagnosis problems
    » watch aggregate benchmark metrics, drill down w/ traceroute
- **Version 1 (human builds, system applies model)**
  - “tell me when condition X is met”
  - human defines alarm conditions, system detects when met
- **Version 2 (system builds, system applies model)**
  - “tell me when something bad happens, and why/where”
  - system defines alarm conditions and detects when met (anomaly detection)
- **Keep human in loop**
  - big red button
  - make model and metrics understandable for human
Questions for current/future work

• Explore techniques for failure inference/diagnosis
  - leverage statistical techniques from Magpie and intrusion detection

• Applicability of statistical techniques from real Internet services to wide-area (need data!!!)

• What is a component?
  - profile Java object time spent and data accesses
    » had undergrads working on this this semester

• Robustness to system flux

• Minimizing code changes to profiled systems

• Handling schema evolution and application-specific metrics
  - XML suggested yesterday

• Using these techniques for intrusion detection
Related work

• Closely related to Magpie (MSR Cambridge)
  - embrace and extend
    » larger, geographically distributed systems
    » explore more models and techniques for change detection

• Part 2 has some relationship to Pinpoint
  - but larger, geographically distributed systems
  - adds latency profiles
  - adds per-component metrics
  - means very different data collection techniques and types of analyses

• Various distributed query processors

• Remote monitoring of instrumented software
Conclusion and status

• Existing data collection/analysis techniques not sufficiently robust and scalable for very large systems in constant flux
  - currently: collect data in per-node logs, aggregate on central node for analysis
  - future: decentralized storage, query, analysis

• Detecting and diagnosing problems is hard
  - currently: collect aggregate metrics (latency, consistency, bandwidth consumed) and per-request metrics (hop-to-hop overlay latencies)
  - future: online data collection, aggregation, analysis; automatically distinguish normal from abnormal component and request statistics (anomaly detection)

• Initial application targets
  - DHTs: Bamboo, Tapestry
  - applications: Seagull, (Palimpsest), other suggestions??