Logging Makes Perfect

Real-world monitoring and visualizations

Itamar Syn-Hershko
http://code972.com
@synhershko
Me?

- Itamar Syn-Hershko / @synhershko
- Blog at http://code972.com
- Long time OSS contributor
- Lucene.NET PMC and lead committer
- Elasticsearch consulting partner
- Building software to catch fraudsters @ Forter
Real time synthesis of social networks, demographics, purchasing power analytics, among many more critical data points.

Understanding the cyber space in the omnichannel era.

Real Time Pattern Recognition.
bae @lanadelcuent  - 20 May 2014
1k more till 10k 😊

Luizzie @LulJozeMonster  - 20 May 2014
@lanadelcuent igi

bae @lanadelcuent  - 20 May 2014
@jozemonster get what

Luizzie @LulJozeMonster  - 20 May 2014
@jozemonster the rts of that tweet

bae @lanadelcuent  - 20 May 2014
@jozemonster me either lol how it went viral 😊😊
DECLINE

Sean Smith, Macbook Air x 3
Sean Smith, Macbook Air x 3

Sean is an officer, serving overseas in Rwanda. His shipping address matches the pattern of diplomat mailboxes for US embassies and his billing address is still linked to his old college accommodation.

How did we know?

BEHAVIORAL DATA
## Different systems, different SLAs

<table>
<thead>
<tr>
<th>TX Processing</th>
<th>Performance</th>
<th>Data Loss</th>
<th>Business Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Low Latency</td>
<td>Nope</td>
<td>Best Effort</td>
</tr>
<tr>
<td>Stream Processing</td>
<td>High Throughput</td>
<td>Best Effort</td>
<td>Best Effort</td>
</tr>
<tr>
<td>Batch Processing</td>
<td>High Volume</td>
<td>Nope</td>
<td>Reconciliation</td>
</tr>
</tbody>
</table>
Forter’s TX Processing API

Transaction details

Decision (approve / decline)

milliseconds
Achieving Low Latencies

For comparison:

• A few network hops between AWS availability zones (~1ms)
• JVM young garbage collection of a few hundred MBs (~20ms)
• Read by primary key from SSD storage (~30ms)
• US West Coast to East Coast network latency (~100ms)
• JVM full garbage collection a few hundred MBs (~500ms-1000ms)
Distributed Systems
Our stack

- nginx (lua)
- expressjs (nodejs)
- Storm (java)
Monitoring and Alerting
What do we monitor?

• Latencies
  – Benchmark tools are useless here
  – Always prefer percentiles, measure the long tail
  – Distinguishing real traffic from synthetic requests

• Availability
  – No non-200 HTTP status codes returned
• High / suspicious metrics
• Errors, exceptions
• Wrong processing (data loss?)
diagnostics

- system tests
- node
- java
- python
- collectd
- cloud-watch
- pingdom

- logstash
- elasticsearch
- kibana

- riemann

- slack
- pagerduty

- Phone notification
- SMS
- Phone Call
- email
• Pingdom probes - low coverage, reliable
• CloudWatch, collectd - fast, no root cause
• Internal probes - better coverage, false alarms
• libbeat and logstash for monitoring 3\textsuperscript{rd} party applications – sometimes noisy, false alarms
• App events (exceptions) - too noisy, root cause
Redundancy of detection sources

.... is important!

- Different concerns
- Varying accuracy and verbosity
- High availability
- Correlation
georg (named after Georg Friedrich Bernhard Riemann) is a nodejs library that acts as an in-process riemann agent. The library can send nodejs specific and application specific metrics to riemann.

https://github.com/forter/georg
Apache Storm Basics
Automatic Latency Reporting: Apache Storm

```java
public class MockSpout implements EventsAware {
    private EventSender es;

    @Override
    public void setEventSender(EventSender es) {
        this.es = es;
    }

    private void foo() {
        es.sendEvent("sent each time foo is called","foo called",1,"mytag1","mytag2");
    }
}
```

https://github.com/forter/riemann-storm-monitor
The ELK Stack

“Shippers”

Queue (Redis)

“Indexer”

{ }

{ }
The ELK Stack + Riemann

“Shippers”

Queue (Redis)
• Detect failures
• Expect different latencies from different components, dynamically
• Tolerate spikes
• High / low watermarks
• Reshape event streams (split, merge, ...)

Done via Riemann – an event stream processor
Riemann

Diagram showing process flow with labels like 'where', 'email', and hosts and services.
; Listen on the local interface over TCP (5555), UDP (5555) & websockets
; (5556)
(let [host "127.0.0.1"]
(tcp-server {:host host})
(udp-server {:host host})
(ws-server {:host host}))

; Expire old events from the index every 5 seconds.
(periodically-expire 5)

(let [index (index)]
; Inbound events will be passed to these streams:
(streams
  (default :ttl 60
    ; Index all events immediately.
    index

    ; Log expired events.
    (expired
      (fn [event] (info "expired" event))))))))
Filter tests using a state machine

index

tagged
apisRegression

changed-state?

where state passed
resolve Pager Duty

where state failed
trigger Pager Duty
Filter tests using a state machine

(tagged "apisRegression"
 (pagerduty-test-dispatch "1234567892ed295d91"))

(defn pagerduty-test-dispatch
  [key]
  (let [pd (pagerduty key)]
    (changed-state {:init "passed"}
      (where (state "passed") {:resolve pd})
      (where (state "failed") {:trigger pd})))
Re-open manually resolved alert

- index
- tagged
- apiRegression
- changed-state?
- where state failed
- where state passed
- by [:host :service]
- resolve Pager Duty
- throttle 1 per 60sec
- trigger Pager Duty
Re-open manually resolved alert

(tagged "apisRegression"
  (pagerduty-test-dispatch "1234567892ed295d91")))

(defn pagerduty-test-dispatch
  [key]
  (let [pd (pagerduty key)]
    (sdo
      (changed-state {:init "passed"}
        (where (state "passed") (:resolve pd)))
      (where (state "failed")
        (by [:host :service]
          (throttle 1 60 (:trigger pd)))))))
# Riemann Events

| **host** | A hostname, e.g. "api1", "foo.com" |
| **service** | e.g. "API port 8000 reqs/sec" |
| **state** | Any string less than 255 bytes, e.g. "ok", "warning", "critical" |
| **time** | The time of the event, in unix epoch seconds |
| **description** | Freeform text |
| **tags** | Freeform list of strings, e.g. ["rate", "fooproduct", "transient"] |
| **metric** | A number associated with this event, e.g. the number of reqs/sec. |
| **ttl** | A floating-point time, in seconds, that this event is considered valid for. Expired states may be removed from the index. |
### Riemann’s Index

<table>
<thead>
<tr>
<th>Key (host + service)</th>
<th>Event</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1-redisfree</td>
<td>{&quot;metric&quot;:&quot;5&quot;}</td>
<td>60</td>
</tr>
<tr>
<td>10.0.0.1-probe1</td>
<td>{&quot;state&quot;:&quot;failed&quot;}</td>
<td>300</td>
</tr>
<tr>
<td>10.0.0.2-probe1</td>
<td>{&quot;state&quot;:&quot;passed&quot;}</td>
<td>300</td>
</tr>
</tbody>
</table>
Riemann Ghosts

• “Expired events”

• Use case: Detect dead services

• Use case: Cron stopped working (e.g. backup)
## Riemann Overloaded

<table>
<thead>
<tr>
<th>key (IP + cookie)</th>
<th>event</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>199.25.1.1-1234</td>
<td>{&quot;state&quot;:&quot;loaded&quot;}</td>
<td>300</td>
</tr>
<tr>
<td>199.25.2.1-4567</td>
<td>{&quot;state&quot;:&quot;downloaded&quot;}</td>
<td>300</td>
</tr>
<tr>
<td>199.25.3.1-8901</td>
<td>{&quot;state&quot;:&quot;loaded&quot;}</td>
<td>300</td>
</tr>
</tbody>
</table>
Something failed, now what?

• Always prefer auto-healing
  – Humans are slow
  – But careful from a domino effect

• Fail fast and automatic failover
  – Code throws exception, logs it and moves forward
  – Fatal exceptions will cause the process to restart (upstart)
  – Auto scaling groups keep enough machines up
  – ELB will phase out dead stacks
  – HTTP fencing (commercial products)
Alert

• When human intervention is required

• Urgent: immediate action
  – PagerDuty (phone calls, SMS, push notifications)

• Low urgency: events are logged and reviewed the morning after
  – PagerDuty (Phone push notifications)
  – Slack
  – Email
Diagnostics
The ELK stack
For real-time analytics and dashboards
The ELK Stack + Riemann
<table>
<thead>
<tr>
<th>Term</th>
<th>Count</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>prod-2015-02-26T1804</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod-2015-02-24T1210</td>
<td></td>
<td></td>
</tr>
<tr>
<td>prod-2015-02-16T1159</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Analyzing sleep patterns 😊

* Times are Israel Day Time
Storm topology visualization & timing
Storm timelines

Latencies – environment: 2015-07-28T0942, jobid: 3933
Alerting via anomaly detection
To conclude…

• Fail fast
  – With back-off
• Log everything
• Alerting via state machines
• By host, service, time
• Alerts are useless without proper debug tools
• Try to self-heal when possible.
Thank you
Questions?

Itamar Syn-Hershko
http://code972.com
@synhershko / @ForterTech