Finding the needle in a Haystack

Ashish Aggarwal / Principal Engineer
Shreya Sharma / Technical Product Manager

@expediahaystack
The Who

expedia group

Bringing the world within reach
The What

Haystack

A resilient, scalable tracing and analysis system
The Why

NOW

MICROSERVICE ARCHITECTURE

- Mainframes 1960s
- Client/server 1970s
- 2 and 3-tiered Systems 1980s
- N-tier Systems 1990s
- Services 2000s
The Why continued
Diagnosing issues across multiple regions is becoming more difficult & time-consuming.
Finding the needle in a Haystack

Source: hikingartist @ FLICKR
The How

Distributed Tracing by Google
The Where and when
The Why again: Build our custom solution

**UUID Support**
- Existing solutions did not support UUID’s as identifiers for Span

**Extensible Platform**
- Ability to build more systems to make use of this immensely useful data stream

**Open Tracing**
- Open Tracing Compliance so the organization is not coupled to a single implementation

**Open Source**
- Opportunity to contribute to the open-source community
Haystack Components

Traces
Trends
Service Graph
Latency Cost
Alerts
Pipes
Traces component Architecture
Traces Waterfall View

<table>
<thead>
<tr>
<th>Start Time</th>
<th>Root</th>
<th>Root Success</th>
<th>Span Count</th>
<th>Svc Duration</th>
<th>Total Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>14 seconds ago</td>
<td>stark-service</td>
<td>snow-1</td>
<td>12</td>
<td>1.260s</td>
<td>3.545s</td>
</tr>
</tbody>
</table>

Traceld: e80965e5-e0c4-4c37-91a7-da79def7597b

Timeline | Latency Cost | Trends | Related Traces
---|--------------|--------|-----------------|
0 | 886ms | 1.772s | 2.659s | 3.545s |

- stark-service
- westeros-service
- tyrell-service
- dragon-service
- blackwater-service
- baratheon-service
- blackwater-service
- westeros-service
- westeros-service
- westeros-service
- westeros-service

Timeline with service names and latency costs.
Trends component Architecture
## Trends component

### Operation

<table>
<thead>
<tr>
<th>Service Name</th>
<th>Count</th>
<th>Duration TP99</th>
<th>Success %</th>
</tr>
</thead>
<tbody>
<tr>
<td>tully-1</td>
<td>total 58.9k</td>
<td>latest 31.453ms</td>
<td>average 99.99%</td>
</tr>
<tr>
<td>tarley-1</td>
<td>total 18.8k</td>
<td>latest 14.53ms</td>
<td>average 70.00%</td>
</tr>
<tr>
<td>cle猜测-1</td>
<td>total 18.8k</td>
<td>latest 31.453ms</td>
<td>average 60.00%</td>
</tr>
<tr>
<td>snow-1</td>
<td>total 15.1k</td>
<td>latest 14.153ms</td>
<td>average 80.00%</td>
</tr>
<tr>
<td>drogo-1</td>
<td>total 15.1k</td>
<td>latest 81.453ms</td>
<td>average 90.00%</td>
</tr>
<tr>
<td>mormont-1</td>
<td>total 5.9k</td>
<td>latest 1.453ms</td>
<td>average 100.00%</td>
</tr>
<tr>
<td>dondarrion-1</td>
<td>total 6.8k</td>
<td>latest 91.453ms</td>
<td>average 10.00%</td>
</tr>
<tr>
<td>grayjoy-1</td>
<td>total 299</td>
<td>latest 14.353ms</td>
<td>average 90.00%</td>
</tr>
</tbody>
</table>
Service Graph component
Network Latency Cost component

| Traceid: e80965e5-e0c4-4c37-91a7-da79def7597b |
|---|---|---|---|
| **Timeline** | **Latency Cost** | **Trends** | **Related Traces** |
| Network time | 996ms (10 measured out of 11 calls) |  |  |
| Network time cross datacenters | 518ms (5 measured out of 6 calls) |  |  |
| Datacenters involved | 6 | **aws us-east-1** | **aws us-east-2** | **aws us-west-1** | **aws us-west-2** | **aws us-west-3** | NA |

**Diagram:Network Latency Cost component**

- **stark-service**: aws us-west-2
  - Avg 92ms, 4 calls
  - 180ms
  - 98ms

- **westeros-service**: aws us-west-2
  - Avg 60ms, 2 calls
  - 121ms
  - 109ms

- **tyrell-service**: NA
  - 180ms

- **dragon-service**: aws us-west-1
  - 98ms

- **baratheon-service**: aws us-west-1
  - 109ms

- **blackwater-service**: aws us-west-1
  - 109ms
Alerts component

Service Name: root-service

- Failure Count: 0
- Duration TP99: 1
- AA Duration: 0

<table>
<thead>
<tr>
<th>Operation</th>
<th>Status</th>
<th>Status Changed</th>
<th>Trend (last 24 hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tarley-1</td>
<td>unhealthy</td>
<td>26 seconds ago at 11:45:00, 15 Aug 18</td>
<td></td>
</tr>
<tr>
<td>tully-1</td>
<td>healthy</td>
<td>23 seconds ago at 11:45:04, 15 Aug 18</td>
<td></td>
</tr>
<tr>
<td>dondarrion-1</td>
<td>healthy</td>
<td>32 seconds ago at 11:44:55, 15 Aug 18</td>
<td></td>
</tr>
</tbody>
</table>

Showing alerts 1 to 3 out of 3
Pipes component

- json-spans
- proto-spans
  - kafka-producer
    - Produces JSON Span objects with flattened tags to Kafka
  - http-poster
    - Posts JSON Span objects with flattened tags to HTTP endpoint
  - firehose-writer
    - Writes JSON Span objects with flattened tags to AWS Firehose
How to onboard / Integrate: Agent

V1 Trace and Blob dispatch Architecture

Your Application

Agent

Traces
Spans

Traces

Read the Config at Bootstrap

Config Service
(Env specific configs for Kinesis/S3)

AWS Service

Amazon Kinesis Streams

GRPC servers
Logical grouping

Haystack Cluster
Haystack flow

Application -> Agent -> Kinesis -> Kafka -> Haystack
Haystack @ Expedia Group

- Multiple brands
- More than few hundred services
- 210k/sec spans ingestion
- 50 node c5.xlarge k8s cluster
- 40 node Kafka cluster
- 50 node c5.xlarge Cassandra
- Tens of ES node cluster
- Support OpenTracing clients in Java and NodeJS
Next steps: Adaptive Alerting

- Adaptive Alerting
- External systems

Metric sources → Anomaly Engine → Model Building → Model Selection & Autotuning → Alerting systems & Automated response systems
References: Open Source Git repositories

https://github.com/ExpediaDotCom/haystack
https://github.com/ExpediaDotCom/haystack-idl
https://github.com/ExpediaDotCom/haystack-commons
https://github.com/ExpediaDotCom/haystack-traces
https://github.com/ExpediaDotCom/haystack-client-java
https://github.com/ExpediaDotCom/haystack-agent
https://github.com/ExpediaDotCom/haystack-trends
https://github.com/ExpediaDotCom/haystack-collector
https://github.com/ExpediaDotCom/haystack-pipes
https://github.com/ExpediaDotCom/haystack-metrics
https://github.com/ExpediaDotCom/haystack-alert-management
https://github.com/ExpediaDotCom/haystack-service-graph