Multi-tenant Pub/Sub Processing for IoT Data Streams

Álvaro Villalba - alvaro.villalba@bsc.es
(22/06/2016)
Introduction

**IoT + Big Data**

Cisco has predicted that we will have over 50 billion connected devices by 2020.

IDC estimates 212 billion "things" globally by the end of 2020. This will include 30.1 billion installed "connected (autonomous) things" in 2020.

7 billion of cellphones active already.

Motivation

- Initiative to produce an advanced Stream Processing Platform for the IoT
- Providing means for data aggregation and processing
- Scalable runtime and multi-tenancy support
- Stream sharing
- Dynamic pipeline
- Added value chain for data
- http://servioticy.com
Wind chill factor:

\[ T_{wc} = 13.12 + 0.6215T_a - 11.37V^{0.16} + 0.3965T_aV^{0.16} \]
ENTITIES & ARCHITECTURE
Entities

WebObject – ServiceObject (SO) – Data Processing Pipe (DPP)
A device has multiple sensors
- Each sensor has multiple dimensions
- Therefore
  - Service Object → Device
  - Stream → Sensor
  - Channel → Dimension
- Streams can be defined as a function of other streams
Service Objects are exposed through a RESTful API.

<table>
<thead>
<tr>
<th>operation</th>
<th>Target URI</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>POST /</td>
<td>Create a new SO posting a JSON document.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>GET /</td>
<td>Retrieve the list of all the SOs created.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>GET /[sold]</td>
<td>Retrieve attributes from the &lt;sold&gt; Service Object.</td>
</tr>
<tr>
<td>Update</td>
<td>PUT /[sold]</td>
<td>Modify the &lt;sold&gt; ServiceObject.</td>
</tr>
<tr>
<td>Delete</td>
<td>DEL /[sold]</td>
<td>Delete the &lt;sold&gt; Service Object.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>GET /[sold]/streams</td>
<td>Retrieve the list of all the SO streams.</td>
</tr>
<tr>
<td>Create</td>
<td>POST /[sold]/streams/[streamId]/subscriptions</td>
<td>Subscribe the SO &lt;sold&gt; to a service.</td>
</tr>
<tr>
<td>Update</td>
<td>PUT /[sold]/streams/[streamId]</td>
<td>Store &lt;sold&gt; data putting a JSON document.</td>
</tr>
<tr>
<td>Retrieve</td>
<td>GET /[sold]/streams/[streamId]</td>
<td>Retrieve the list of all the data of &lt;sold&gt; Service Object.</td>
</tr>
<tr>
<td>lastUpdate</td>
<td>GET /[sold]/streams/[streamId]/lastUpdate</td>
<td>Retrieve the last piece of data generated by a &lt;sold&gt; Service Object.</td>
</tr>
</tbody>
</table>

+ actuations + searches
Data-Centric Architecture: technologies

- Apache Storm
- Kafka
- Couchbase
- Elasticsearch
- Node.js
- Apollo
- Stomp

Stream-Processing Topology

Front-End

Back-End
Storm topology

Kafka → Input → Prepare → Elastic Search → Couch Base

External Subscriptions:
- PubSub dispatch

Internal Subscriptions:
- Stream dispatch → Stream process
DATA PROCESSING PIPELINES
Data processing pipelines features

- Publisher-Subscriber
- User code
- Expansion on-the-fly
- Reactive computation
- Programming model
- Authorization policies
- Dynamic subscriptions

Wind chill factor

\[ T_{\text{ch}} = 13.12 + 0.6215T_a - 11.37V^{0.16} + 0.3965T_a V^{0.16} \]
Operations

- Index and query
- Filter
- Transform
- Compose
- Aggregate
- Group

Pub/sub communication
Data processing pipelines stages

1. Receive update
2. Subscriber dispatching
3. Additional data fetching
4. Transformation & filtering
5. Store & emit
Dynamic subscriptions

- For groups and aggregations
- Based on stream characterizations
Main design principles

**A stream is sequential**
- One update after another in strict order

**Several streams with the same sources can run in parallel**

**Priority in current state**

**Ideally the computation is faster than the interval between updates in a stream**
- Some updates are discarded otherwise from the most active streams

**Not blocking.**
- If there is an error processing an update and new updates are waiting to be processed, discard the old one
  - Timeouts
  - No blocking ‘zip’ to compose streams
Non-blocking composition example

[Diagram showing components and flows labeled SU 2, SU 3, SU 1, SU 4, and STORAGE]
Strict order and cycles control

- Input update must be newer than last output update
- Output update timestamp is the higher from the inputs
- Avoids cycles and controls strict order
DPP generator

- Python scripts collection
  - NetworkX module

- Configurable:
  - Number of independent topologies
  - Number of SOs
  - Number of streams (initial and composite)
  - Number of channels
  - Number of operands per composite stream
  - Distribution used to select operands
  - Random operands

- All numeric values can be specified as a statistical distribution

- Measuring DPP stages latencies
  - Input and Output

- Measuring source to sink latencies

https://github.com/servioticy/servioticy-dispatcher-benchmark
Significative graphs

Measuring latency impact on:

- Length
- In-degree
- Out-degree
First try – Random graphs with cycles

- Big graphs with cycles (+1000 DPPs)
- High latency
- Most computations were noise
  - Updates that did not reach the sink DPPs
Second try – Random DAGs

- DAGs with +100 DPPs
- Still high latency
- Most of noise was still there
Third try – Random ‘merged trees’

- Graphs with one source-sink path
- Sinks shared between sources
- +100 DPPs
- Without noise
Stages latency

- 149 edges
- 42 nodes

- 523 edges
- 80 nodes
Significative graphs latencies

![Graph showing execution time vs. number of streams](image-url)
Completed work

ServIoTicy¹

- Multi-tenant and centralized IoT platform
- Event-driven and non-blocking


1. http://servioticy.com
Thank you!

For further information please contact me
alvaro.villalba@bsc.es