Prototyping a New Ingest Pipeline System for SDP

Drew Devereux
16 February 2018
Problem definition
Problem definition

Smaller data
Many data
Lower latency
Lower value data

Larger data
Fewer data
High latency
Critical data
Problem definition

ingest
Problem definition

• Define the requirements of the ingest stage
• Determine upon an architectural style
• Design, implement and test a prototype
Requirements

- High throughput
  - Not necessarily real-time latencies, but must be able to keep up
  - We need ingest to finish soon enough after the observation that the rest of the system has time to do its processing
Requirements

• High throughput
  • Not necessarily real-time latencies, but must be able to keep up
  • We need ingest to finish soon enough after the observation that the rest of the system has time to do its processing

• Available / reliable / fault-tolerant
  • Dropping an observation because a buffer is full – no problem
  • Crashing because a buffer is full – big problem
Requirements

• High throughput
  • Not necessarily real-time latencies, but must be able to keep up
  • We need ingest to finish soon enough after the observation that the rest of the system has time to do its processing

• Available / reliable / fault-tolerant
  • Dropping an observation because a buffer is full – no problem
  • Crashing because a buffer is full – big problem

• Configurable / composable
  • The configuration of the observation determines the configuration of the processing system.
Requirements

• High throughput
  • Not necessarily real-time latencies, but must be able to keep up
  • We need ingest to finish soon enough after the observation that the rest of the system has time to do its processing

• Available / reliable / fault-tolerant
  • Dropping an observation because a buffer is full – no problem
  • Crashing because a buffer is full – big problem

• Configurable / composable
  • The configuration of the observation determines the configuration of the processing system.

• Manageable
  • Ability to monitor the system and intervene if necessary
Architectural approach

• Service oriented architecture (SOA)
  • “Services” are independent, isolated components that interact only via a communication protocol
  • Communication is usually asynchronous to maximise decoupling e.g. request/response, publish/subscribe
  • Isolation of services makes SOA systems configurable, reliable, fault-tolerant, deployable.
Architectural approach

• Service oriented architecture (SOA)
  • “Services” are independent, isolated components that interact only via a communication protocol
  • Communication is usually asynchronous to maximise decoupling e.g. request/response, publish/subscribe
  • Isolation of services makes SOA systems configurable, reliable, fault-tolerant, deployable.

• Microservices
  • Each service does one thing well
Architectural approach

• Service oriented architecture (SOA)
  • “Services” are independent, isolated components that interact only via a communication protocol
  • Communication is usually asynchronous to maximise decoupling e.g. request/response, publish/subscribe
  • Isolation of services makes SOA systems configurable, reliable, fault-tolerant, deployable.

• Microservices
  • Each service does one thing well

• Streaming
Architectural approach

Messaging System abstraction layer

Messaging System
## Streaming engines

<table>
<thead>
<tr>
<th>Engine</th>
<th>Fluent</th>
<th>Mif</th>
<th>Sparkpump</th>
<th>Apex</th>
<th>Kafka Streams</th>
<th>Spark Streaming</th>
<th>Storm</th>
<th>Storm + Trident</th>
<th>Samza</th>
<th>Flink</th>
<th>Ignite Streaming</th>
<th>Beam [G20 DataFlow]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current version</td>
<td>1.0</td>
<td>0.1</td>
<td>3.0</td>
<td>0.0</td>
<td>0.3.1</td>
<td>0.10.0</td>
<td>1.6</td>
<td>0.12.0</td>
<td>1.0</td>
<td>1.5.3</td>
<td>1.0</td>
<td>1.5.0</td>
</tr>
<tr>
<td>Events size</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
<td>simple</td>
</tr>
<tr>
<td>Contributors</td>
<td>26</td>
<td>67</td>
<td>10</td>
<td>53</td>
<td>180</td>
<td>039</td>
<td>207</td>
<td>48</td>
<td>159</td>
<td>56</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Main backers</td>
<td>Apache</td>
<td>Cloudera</td>
<td>Hortonworks</td>
<td>LinkedIn</td>
<td>Google</td>
<td>LinkedIn</td>
<td>Google</td>
<td>LinkedIn</td>
<td>Google</td>
<td>LinkedIn</td>
<td>Google</td>
<td>LinkedIn</td>
</tr>
<tr>
<td>Delivery guarantees</td>
<td>at least once</td>
<td>exactly once</td>
<td>exactly once</td>
<td>at least once</td>
<td>exactly once</td>
<td>at least once</td>
<td>exactly once</td>
<td>at least once</td>
<td>exactly once</td>
<td>at least once</td>
<td>exactly once</td>
<td>exactly once</td>
</tr>
<tr>
<td>State management</td>
<td>transactional updates</td>
<td>local and distributed snapshots</td>
<td>checkpoints</td>
<td>local and distributed snapshots</td>
<td>checkpoints</td>
<td>local snapshots</td>
<td>distributed snapshots</td>
<td>local snapshots</td>
<td>distributed snapshots</td>
<td>checkpoints</td>
<td>transactional updates</td>
<td></td>
</tr>
<tr>
<td>Fault tolerance</td>
<td>yes (with idle channel only)</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Out-of-order processing</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Event prioritization</td>
<td>no</td>
<td>yes</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
<td>programmable</td>
</tr>
<tr>
<td>Windowing</td>
<td>no</td>
<td>no</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
<td>time-based</td>
</tr>
<tr>
<td>Back-pressure</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>N/A</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Primary abstraction</td>
<td>Event</td>
<td>Flow</td>
<td>Message</td>
<td>Tuple</td>
<td>KafkaStream</td>
<td>Storm</td>
<td>Tuple</td>
<td>Storm</td>
<td>Tuple</td>
<td>Storm</td>
<td>Stream</td>
<td>Storm</td>
</tr>
<tr>
<td>Data flow</td>
<td>agent</td>
<td>flow (process group)</td>
<td>streaming application</td>
<td>streaming application</td>
<td>streaming application</td>
<td>topology</td>
<td>application</td>
<td>topology</td>
<td>topology</td>
<td>application</td>
<td>pipeline</td>
<td></td>
</tr>
<tr>
<td>Latency</td>
<td>slow</td>
<td>fast</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
<td>very low</td>
</tr>
<tr>
<td>Resource management</td>
<td>native</td>
<td>native</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
<td>WARN</td>
</tr>
<tr>
<td>Autoscaling</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>In-flight modifications</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Primarily written in</td>
<td>Java</td>
<td>Scala</td>
<td>Java</td>
<td>Java</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
<td>Scala</td>
</tr>
<tr>
<td>API languages</td>
<td>REST (HTTP)</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Java</td>
<td>Scala</td>
<td>Java</td>
<td>Python</td>
<td>Java</td>
<td>Scala</td>
<td>Java</td>
<td>Java</td>
</tr>
<tr>
<td>Notable owners</td>
<td>Maestro Sherman, Simple,Inc</td>
<td>Bondi</td>
<td>N/A</td>
<td>Intel</td>
<td>Let's Talk</td>
<td>Capital One</td>
<td>N/A</td>
<td>Kello</td>
<td>Netflix, Akamai</td>
<td>Uber</td>
<td>Netflix, Akamai</td>
<td>Netflix</td>
</tr>
</tbody>
</table>

---

14 | Prototyping a New Ingest Pipeline System for SDP | Drew Devereux
Architectural approach
Elements of the system

• Messaging system
  • publish / subscribe
  • Multiple messaging systems

• Service container
  • Instantiates and runs a service

• Services for
  • Data processing
  • Monitoring and control
  • Orchestration
Architecture

Prototyping a New Ingest Pipeline System for SDP | Drew Devereux

[Diagram showing the architecture of the messaging system with various services and layers]
Status and future work

• We have a working design
• We have a prototype, with services that run, process data and communicate over a messaging system
• We have basic monitoring and control
• Configuration-driven orchestration is next
Thank you

CSIRO Astronomy and Space Science
Drew Devereux
Research Scientist

+61 8 6436 8878
drew.devereux@csiro.au

CSIRO Astronomy and Space Science
JC Guzman
ATNF Software & Computing Group Leader

+61 8 6436 8569
juan.guzman@csiro.au