Streaming
Why should I care?

Christian Trebing
Blue Yonder GmbH
@ctrebing
Agenda

Motivation
Streaming Intro
Implementation
Challenges
Data Processing
- The Monolith

- Data Input
- Data Validation
- Machine Learning
- Data Output

THE Database
Pain

Several teams are developing this application

• Customer desperately wants new feature in machine learning

• But the data validation team is in the midst of refactoring their database structure (‘will be finished in two weeks’) 

• So you wait…
Could Microservices Help?

Great:
• No Dependency on single state
• Independent Development
• Independent Upgrades

Difficult:
• Too much data to transfer
• Too much data to store in each service
Are There Other Possibilities?
Streaming Intro
### Databases and Streams

Same information in table and stream:

<table>
<thead>
<tr>
<th>A=1</th>
<th>B=5</th>
<th>C=3</th>
<th>A=8</th>
<th>A=4</th>
<th>C=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td>A: 1</td>
<td>A: 1</td>
<td>A: 8</td>
<td>A: 4</td>
<td>A: 4</td>
</tr>
<tr>
<td>B: 5</td>
<td>B: 5</td>
<td>B: 5</td>
<td>B: 5</td>
<td>B: 5</td>
<td>B: 5</td>
</tr>
<tr>
<td>C: 3</td>
<td>C: 3</td>
<td>C: 3</td>
<td>C: 3</td>
<td>C: 2</td>
<td></td>
</tr>
</tbody>
</table>
Why does it matter?

Different services can be in different states
  - Each service can consume the stream in its own speed
  - One service can be updated while the other runs

<table>
<thead>
<tr>
<th></th>
<th>A=1</th>
<th>B=5</th>
<th>C=3</th>
<th>A=8</th>
<th>A=4</th>
<th>C=2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td></td>
<td></td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>5</td>
<td></td>
<td></td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

Service 1 on index 3
Service 2 on index 5
Partitioned Streams

Sales stream, partitioned by location
Each partition could be handled by diff. processors

<table>
<thead>
<tr>
<th>Location: Rimini</th>
<th>Location: Rimini</th>
<th>Location: Rimini</th>
<th>Location: Rimini</th>
</tr>
</thead>
<tbody>
<tr>
<td>product: Spaghetti</td>
<td>product: Ravioli</td>
<td>product: Pizza</td>
<td>product: Spaghetti</td>
</tr>
<tr>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-11</td>
<td>sales_date: 2017-07-10</td>
</tr>
<tr>
<td>quantity: 5</td>
<td>quantity: 8</td>
<td>quantity: 1</td>
<td>quantity: 7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location: Bilbao</th>
<th>Location: Bilbao</th>
<th>Location: Bilbao</th>
</tr>
</thead>
<tbody>
<tr>
<td>product: Pizza</td>
<td>product: Ravioli</td>
<td>product: Spaghetti</td>
</tr>
<tr>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-11</td>
<td>sales_date: 2017-07-11</td>
</tr>
<tr>
<td>quantity: 3</td>
<td>quantity: 9</td>
<td>quantity: 5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location: Karlsruhe</th>
<th>Location: Karlsruhe</th>
<th>Location: Karlsruhe</th>
<th>Location: Karlsruhe</th>
</tr>
</thead>
<tbody>
<tr>
<td>product: Pizza</td>
<td>product: Ravioli</td>
<td>product: Ravioli</td>
<td>product: Spaghetti</td>
</tr>
<tr>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-10</td>
<td>sales_date: 2017-07-11</td>
</tr>
<tr>
<td>quantity: 8</td>
<td>quantity: 3</td>
<td>quantity: 7</td>
<td>quantity: 7</td>
</tr>
</tbody>
</table>
Data Processing
- With Streaming

Data Input → Data → Streaming Platform → Data Validation → Data

Data Output → Data → Streaming Platform → Machine Learning → Data
What did we gain?

Independent Development
Independent Upgrade
Scalability
Did we throw out databases completely?
• Let’s see…
Is it magic?

No, it’s a tradeoff:

- A database is so powerful: ACID guarantees, SQL language. You can do nearly everything
  - This comes at a price
    - Dependency on single state
    - Scaling is hard

So let’s more think what we really need
What do we loose?

<table>
<thead>
<tr>
<th>Database</th>
<th>Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACID</td>
<td>Ordering on stream partition</td>
</tr>
<tr>
<td>SQL Queries</td>
<td>Service is responsible of keeping its state</td>
</tr>
</tbody>
</table>

You have to decide whether you can live with that.
Implementation
Apache Kafka

![Diagram of Apache Kafka architecture](image-url)
Kafka Clients in Python

• pykafka, python-kafka, confluent-kafka-client
• Nice comparison has been done here:
  • http://activisiongamescience.github.io/2016/06/15/Kafka-Client-Benchmarking/
• Most performant currently is confluent-kafka-client
  • Uses the c library librdkafka
import json
from confluent_kafka import Producer

p = Producer({"bootstrap.servers": "localhost:9092"})
data = {
    "location": "Rimini", "product": "Ravioli",
    "sales_date": "2017-07-10", "quantity": 5
}
p.produce('sales_input', json.dumps(data))
p.flush()
from confluent_kafka import Consumer, KafkaError
import uuid

c = Consumer({'bootstrap.servers': 'localhost:9092', 'group.id': uuid.uuid1(),
               'default.topic.config': {'auto.offset.reset': 'smallest'}})
c.subscribe(['sales_input'])
running = True

while running:
    msg = c.poll()
    if not msg.error():
        print('Received message: %s' % msg.value().decode('utf-8'))
    elif msg.error().code() != KafkaError._PARTITION_EOF:
        print(msg.error())
        running = False
    c.close()
Apache Avro
Data Serialization, Enabling Schema Evolution

Clearly defined schema with:
• Schema evolution
• Schema registry

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>location</td>
</tr>
<tr>
<td>string</td>
<td>product</td>
</tr>
<tr>
<td>string</td>
<td>sales_date</td>
</tr>
<tr>
<td>int</td>
<td>quantity</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Field Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>string</td>
<td>location</td>
</tr>
<tr>
<td>string</td>
<td>product</td>
</tr>
<tr>
<td>string</td>
<td>sales_date</td>
</tr>
<tr>
<td>int</td>
<td>quantity</td>
</tr>
<tr>
<td>int, default=0</td>
<td>delivery_id</td>
</tr>
</tbody>
</table>
import json
from confluent_kafka import avro

value_schema = avro.loads(json.dumps({
    "name": "sales_input_value",
    "type": "record",
    "fields": [
        {
          "name": "location", "type": ["string"]
        },
        {
          "name": "product", "type": ["string"]
        },
        {
          "name": "sales_date", "type": ["string"]
        },
        {
          "name": "quantity", "type": ["int"]
        }
    ]
}))
from confluent_kafka import avro
from confluent_kafka.avro import AvroProducer
from schema import key_schema, value_schema

avroProducer = AvroProducer({
    'bootstrap.servers': 'localhost:9092',
    'schema.registry.url': 'http://localhost:8081',
    default_key_schema=key_schema, default_value_schema=value_schema)

key = {'location': 'Rimini'}
value = {
    'location': 'Rimini',
    'product': 'Ravioli',
    'sales_date': '2017-07-10',
    'quantity': 5
}

avroProducer.produce(topic='sales_input', value=value, key=key)
avroProducer.flush()
```
import uuid
from confluent_kafka import KafkaError
from confluent_kafka.avro import AvroConsumer
from confluent_kafka.avro.serializer import SerializerError

c = AvroConsumer(
    {'bootstrap.servers': 'localhost:9092', 'group.id': uuid.uuid1(),
     'schema.registry.url': 'http://localhost:8081',
     'default.topic.config': {'auto.offset.reset': 'smallest'}})

c.subscribe(['sales_input'])
running = True
while running:
    try:
        msg = c.poll(10)
        if msg:
            if not msg.error():
                print(msg.value())
            else:
                if msg.error().code() != KafkaError._PARTITION_EOF:
                    print(msg.error())
                    running = False
                else:
                    print("Message deserialization failed for \%s: \%s" % (msg, e))
                    running = False
    except SerializerError as e:
        print("Message deserialization failed for \%s: \%s" % (msg, e))
        running = False

    c.close()
```
Example: Data Validation

Separate valid and invalid sales records

```python
consumer.subscribe(['sales_input'])

while True:
    msg = self.consumer.poll(1)
    if msg:
        key = msg.key()
        value = msg.value()
        if sales_record_valid(value):
            producer.produce(topic='sales_validated', value=value, key=key)
        else:
            producer.produce(topic='sales_error', value=value, key=key)
```
Additional Processors

Need to evolve your application:
- Add processors for evaluation topics
- Try new variant of validation logic

database
- remember processing state, same for each processor

streaming
- offset in each processor, can work independently
Challenge
- Machine Learning Still Batch
How to get the input data?

Remember: There is no possibility to query a stream. Somewhere all this data needs to be. Options:
- Memory of the service
- Serving database
- Blob store

Yes, that’s duplication. We’ll have to live with it.
Write Path / Read Path

data validation  THE database  machine learning query  machine learning

write path

write path

data validation  machine learning query  Blob store  machine learning

read path

read path
Machine Learning Input Data

- locations_validated
- sales_validated
- products_validated

Join

Append to file
Challenge
- State in Processors
State - Nightmare of every distributed systems engineer

Streaming: Data just rushes through

Why do we need state?
• Time window processing
• Data you want to join with

Formerly, the database did it for you
State - Some Challenges?

Failure of a processor

Scaling
State in Stream Processors
- Possible Solutions

- Just keep in memory. Reprocess stream to warm up
- Each processor to keep its own db
- Save condensed in stream
- Get it from other service

Frameworks exist in other languages:
- for example: Kafka Streams, Apache Samza

Up to yesterday: none in python. Then heard about

https://github.com/wintoncode/winton-kafka-streams
Summary

- You have more options for your data processing applications than you might have thought
- As always, there are some tradeoffs
- You know the challenges
Questions?
Blue Yonder
Best decisions, delivered daily