Databases and Stream Processing: A Future of Consolidation
Marc Andreessen: Software is Eating the World
Weak Form
Companies are USING MORE SOFTWARE

Strong Form
Companies are BECOMING SOFTWARE
Loan Application Using Software

1. BORROWER
2. APPLICATION FORM
3. CREDIT OFFICER
4. RISK OFFICER
5. LOAN OFFICER
6. APPROVE
   DENY
Using Software:
Classic Three-Tier Architecture

USER → UI → SERVICE → DATABASE
Becoming Software: Services Talking To Each Other With APIs
REQUESTING A RIDE
Evolution of software systems

Monolith

Distributed Monolith

Microservices

Event-Driven Microservices

UI

App

App

App

Service

Service

Service

Service

Service

Service

Service

Service

Service

Kafka

Increasing Complexity
THE USER OF THE SOFTWARE IS MORE SOFTWARE
What does this mean for databases?
We have hundreds of databases...
FUNDAMENTAL ASSUMPTION:
DATA IS PASSIVE
Databases are designed to help you!
Unless there is a user and UI waiting, why should it be synchronous?
The Alternative: Event Streams
Stream Processors are built for Asynchronicity
Stream Processors have a different interaction model

**TRADITIONAL DATABASE**
- Active Query: `SELECT * FROM DB_TABLE`
- Passive Data: DB Table

**EVENT STREAM PROCESSING**
- Active Data: Event Stream
- Passive Query: `CREATE TABLE AS SELECT * FROM EVENT_STREAM`
Streams or Tables?
An Event
records the fact that something happened

A good was sold
An invoice was issued
A payment was made
A new customer registered
Events are state changes, they carry intent

State:

Bob works at Google

Event:

Bob moved from Google to Amazon
Streams record exactly what happened

Where you have been vs. Where you are now

Payments you made vs. Your account balance

Tables current state
Streams
A sequence of moves

1. e4 e5
2. Nf3 Nc6
3. Bc4 Bc5
4. d3 Nf6
5. Nbd2

Tables
Position of each piece
Streams = INSERT only
Immutable, append-only

Tables = INSERT, UPDATE, DELETE
Mutable, Primary Key
A stream can be considered as an immutable, append-only table.
Stream Processors Communicate Through Streams

INPUT STREAMS

SP

OUTPUT STREAMS
But internally they use tables

CREATE TABLE credit_scores AS
SELECT user, updateScore(p.amount)...

<table>
<thead>
<tr>
<th>USER</th>
<th>PAYMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>JAY</td>
<td>42</td>
</tr>
<tr>
<td>SUE</td>
<td>18</td>
</tr>
<tr>
<td>FRED</td>
<td>65</td>
</tr>
</tbody>
</table>

Payments Stream

Credit Score Table

Credit Score Stream
Streams record history

Tables represent state

projection
(Group By Key, SUM, COUNT)

Duality

table changes

*See Streams and Tables: Two Sides of the Same Coin, M. Sax et al., BIRTE '18
Similar to a materialized view in a database
Joins
Joining a stream with a table

Orders

Customers

Lookup Customer

Table of Customers (with Primary Key)
Joining two streams

orders.join(payments)
Joining two streams

orders.join(payments)
Joining two streams

orders.join(payments)
Joining two streams

orders.join(payments)
Joining two streams

Bob's Order

Jill's Order

Key-value store

Bob's Payment

Jill's Payment
Joining two streams

Bob's Order

Jill's Order

Key-value store

Bob's Payment

Jill's Payment
Joining two streams
Joining two streams
Joining two streams
Joining two streams
Joining two streams
Joining two streams
Streams represent history $\rightarrow$ Cartesian Product

<table>
<thead>
<tr>
<th>Orders Stream</th>
<th>Payments Stream</th>
</tr>
</thead>
<tbody>
<tr>
<td>101 Boots</td>
<td>101 $50</td>
</tr>
<tr>
<td>200 Hat</td>
<td>200 $10</td>
</tr>
<tr>
<td>101 Boots2</td>
<td>105 $3</td>
</tr>
<tr>
<td>105 Pants</td>
<td>200 $12</td>
</tr>
<tr>
<td>200 Hat2</td>
<td>101 $60</td>
</tr>
</tbody>
</table>

Join Output (Stream)
Joining Streams to Streams

Orders Stream

<table>
<thead>
<tr>
<th>ID</th>
<th>Item</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>Boots</td>
</tr>
<tr>
<td>200</td>
<td>Hat</td>
</tr>
<tr>
<td>101</td>
<td>Boots2</td>
</tr>
<tr>
<td>105</td>
<td>Pants</td>
</tr>
<tr>
<td>200</td>
<td>Hat2</td>
</tr>
</tbody>
</table>

Payments Stream

<table>
<thead>
<tr>
<th>ID</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>$50</td>
</tr>
<tr>
<td>200</td>
<td>$10</td>
</tr>
<tr>
<td>105</td>
<td>$3</td>
</tr>
<tr>
<td>200</td>
<td>$12</td>
</tr>
<tr>
<td>101</td>
<td>$60</td>
</tr>
</tbody>
</table>

Join Output (Stream)

Use time window
Tools for correlating recent events in time
More advanced temporal functions

Orders → Page Visits

Session

Join Output (Stream)
Late and out-of-order data

Orders → Page Visits

Join Output (Stream)

Window 1

Window 2
Stream processors provide tools that handle **asynchronicity**, leverage **time** and focus on ‘now’.
Data Placement
Layered storage model

Storage (Kafka)

‘Caching’ in streaming layer

from stream’s P2

from table’s P2

Stream Processor

read via network
Partitioned Data (Fact-Fact joins)

Storage (Kafka)

- P1
- P2
- P3
- P4

Partitioned KTable / TABLE

- SP 1: 2 GB
- SP 2: 3 GB
- SP 3: 5 GB
- SP 4: 2 GB

Storage (Kafka)
Broadcast Data (Fact-Dimension Joins)

GlobalKTable

P1

P2

P3

P4

Stream Task 1

Stream Task 2

Stream Task 3

Stream Task 4

GlobalKTable

2 + 3 + 5 + 2 = 12 GB

12 GB

12 GB

12 GB

12 GB
Architecturally there are parallels e.g. Data Warehousing.
Interaction Model
Stream Processors Continuously Process Input to Output
TRADITIONAL DATABASE

Active Query
SELECT * FROM DB_TABLE

Passive Data
DB Table

EVENT STREAM PROCESSING

Active Data
CREATE TABLE AS SELECT * FROM EVENT_STREAM

Passive Query
Event Stream
Databases are Pull Queries

What is Ben’s credit score now?

695

Stream Processors are Push Queries

Payments

Ben’s credit score is 670
Ben’s credit score is 710
Ben’s credit score is 695

...
Hybrid stream processors provide both interaction models.
Unified Model For:

1. The **Asynchronous** and the **Synchronous**
2. Interaction with **Active** or **Passive** Data
Unified interaction model

Earliest to now

The Past

Now

The Future

Standard Database Query
Unified interaction model

The Past

Now

The Future

Standard Stream Processing Query

Now to forever
Unified interaction model

Earliest to forever

‘Dashboard query’

The Past
Now
The Future
Unified Interaction Model

- Earliest to now
- Earliest to forever
- Now to forever
PUSH

SELECT user, credit_score
FROM orders
WHERE ROWKEY = 'bob'
EMIT CHANGES;

PULL

SELECT user, credit_score
FROM orders
WHERE ROWKEY = 'bob';
Asynchronous => Pipelines

Transactions

Joins/aggregation/time-handling
Other important variants

- Stream processors are often programming frameworks today
  - Storm
  - Flink
  - Kafka Streams
- Today we have active databases that include change streams:
  - Mongo
  - Couchbase
  - RethinkDB
As Software Eats the World
THE USER OF THE SOFTWARE IS MORE SOFTWARE
We need
Asynchronous + Synchronous
Active + Passive
We still need all of these
So is the traditional perception of “a database” enough?
Ben Stopford
Confluent
@benstopford
ben@confluent.io