Predictable stream processing with the Trevor auto-scaler

Manu Bansal

with Eyal Cidon, Sachin Katti, Christos Kozyrakis

Stanford Platform Lab Retreat, June 9, 2017, Aptos, CA
Real-time stream processing pipelines

**Real-time**
- < 1ms

*Low latency*

**Wireless data planes**
- Multi-core DSPs
- Multi-media transcoders
- Financial trading

**Wireless control planes**
- Cluster of machines
- Ad analytics pipelines
- Trending topics pipeline

**Near real-time**
- 100ms-10s

*High throughput*
How do you build a software-defined wireless data plane?
The timing problem

How many cores?
Local memory or shared?
How much parallelism?
What runs where?
Layout resilient to change?

10s of microseconds
Atomix for real-time stream processing

Real-time < 1ms

Wireless data planes
  - Multi-core DSPs
  - Multi-media transcoders
  - Financial trading

Atomix
Model-based programming framework for predictable timing
How do you build a software-defined wireless control plane?
The provisioning problem

- Parse JSON Events
- Filter Events
- Join with User IDs
- Compute cell load

How many worker machines?
What kind of worker machines?
How much parallelism?
What runs where?
Layout resilient to change?

User: ...
Cell: ...
Thp: ...

User: ...
Cell: ...
Thp: ...

User: ...
Cell: ...
Thp: ...

10k – 1m events/min
Trevor for near real-time stream processing

Near real-time 100ms-10s

Wireless control planes
Cluster of machines
Ad analytics pipelines
Trending topics pipeline

Trevor
Model-based auto-scaler for predictable throughput
Streaming analytics pipeline for 3000 cells of a metro area
Load balancing, interference management, content delivery...
New event types, vendor infrastructure upgrades
Increasing network scale, new geographies
Logic improvements and updates
Migrating clusters (cloud to on-prem dev to on-prem prod)

Pipeline, infrastructure, network scale changes every month
The provisioning problem

Parse JSON Events → Filter Events → Join with User IDs → Compute cell load

How many worker machines?
What kind of worker machines?
How much parallelism?
What runs where?
**Layout resilient to change?**

User: ...
Cell: ...
Thp: ...

User: ...
Cell: ...
Thp: ...

User: ...
Cell: ...
Thp: ...

10k – 1m events/min
No performance models!
All trial and error...
Workflow with performance predictions

1. Design time performance predictions
2. Performance meets target?
3. Deployment (efficient and meets target performance)

Workload

Software implementation

Component profiles

Candidate layout

DEV & CHANGE EFFORT

LAYOUT SEARCH
A simple word-count flow graph

A word-count flow graph with two components

<table>
<thead>
<tr>
<th>Word</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;word1&gt;</td>
<td>&lt;count1&gt;</td>
</tr>
<tr>
<td>&lt;word2&gt;</td>
<td>&lt;count2&gt;</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Components, instances, containers

1. **Components** (nodes)
   - Parallelism
     - Producers: 2
     - Consumers: 3

2. **Instances** (processes)
   - Packing
     - Containers: 3
     - Sizes: 3CPU/4GB
     - Packing: RR

3. **Containers** (resource sandboxes)
   - CPU: 3
   - Mem: 4GB
   - Packing: RR

Fields Grouping
- (<word>, 1)

Data is shuffled with consistent hashing of keys

<table>
<thead>
<tr>
<th>&lt;word1&gt;</th>
<th>&lt;count1&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;word3&gt;</td>
<td>&lt;count3&gt;</td>
</tr>
<tr>
<td>&lt;word5&gt;</td>
<td>&lt;count5&gt;</td>
</tr>
<tr>
<td>&lt;word2&gt;</td>
<td>&lt;count2&gt;</td>
</tr>
<tr>
<td>&lt;word4&gt;</td>
<td>&lt;count4&gt;</td>
</tr>
</tbody>
</table>
Performance is sensitive to config

- Parallelism?
  - Too little $\rightarrow$ bottlenecks
  - Too much $\rightarrow$ degradation

- Containers?
  - Too few $\rightarrow$ shuffle bottleneck
  - Too many $\rightarrow$ over-provisioning

- Packing?
  - Locality
  - Shuffle degree
  - Load distribution
Achieved 30x scaling by tuning Spark Streaming wireless analytics pipeline

24 cores x 8 workers \( \rightarrow \) 16 cores x 4 workers

(192 cores) \( \rightarrow \) (64 cores)
Reactive auto-scaling is too slow

Dhallion* takes an hour to reach steady state on a 3-stage topology!

* Dhallion [Floratou et al., to appear in VLDB ’17]
How do you build a predictor?
Black-box modeling too slow

• Naïve approach: black-box modeling
  • Not scalable – too many configurations
Modular component cost models
Models do not compose (yet)

Measured: 965k tup/sec
Predicted: 1320k tup/sec

A flowgraph could bottleneck on data shuffle
Modeling the stream manager
Models still do not compose (yet)

Measured: 965k tup/sec
Predicted: 1320k tup/sec

Container-local and cross-container flows cost different
Modeling shuffle data rates

Container-local and cross-container flows cost different
We solve shuffles with a linear program

Shuffle operator behavior can be expressed as constraints

r11 = r12
r21 = r22
We solve shuffles with a linear program

Tying to shuffle constraints
\[ r = r_{11} + r_{12} \]
\[ s = r_{21} + r_{22} \]

Tying local flow constraints
\[ e_{11} = r_{11} \]
\[ e_{22} = r_{22} \]

Shuffle operator cost can be modeled with egress and ingress nodes
Models composed with shuffle a/c

Measured: 965k tup/sec
Predicted: 1050k tup/sec!
We can predict instance scaling
We can predict container scaling
Balanced-container provisioning

Key idea: Schedule rate-matched containers for *edges*

\[
\text{tuplerate}(n_P \text{ instances of } P) = \text{tuplerate}(n_Q \text{ instances of } Q) = 0.25 \times \text{tuplerate}(1 \text{ instance of } S)
\]

* Should we call them “molecules”? Bids for naming the Trevor project will also be accepted in Q&A.
Balanced-container provisioning

Turns a bin-packing problem into a waterfilling problem
Permits divide-and-conquer, so scales to large topologies
Predictive provisioning is near-optimal

We can compute a near-optimal steady-state provisioning in < 1s
Our near-optimal provisioning performs as predicted
• Model cost of computing user operations and shuffling data
• Solve for local shuffle rates and distributed shuffle rates
• Find scaling using a balanced-container algorithm

• Key learnings
  • Possible to predict performance of streaming pipelines
  • Predictions bring significant gain in provisioning effort
QUESTIONS? COMMENTS?