Autoscaling Streaming Jobs with ML

Ioannis Prapas

Ankush Sharma

Sokratis Papadopoulos

Supervised by Ankit Chaudhary

Agenda

- Goal – Problem – Solution
- Architecture
- RL model
- Experiments
- Conclusion

Goal

Use **minimum resources** to achieve
optimal job **throughput** and **latency**

In other words, we want parallelism level to
dynamically follow the data generation rate

Problem

- Parallelism level of a job is set manually
  - Over-provision: Use more resources than needed most of the time
  - Under-provision: Accept degraded performance
- Data streams processing jobs require different parallelism across time

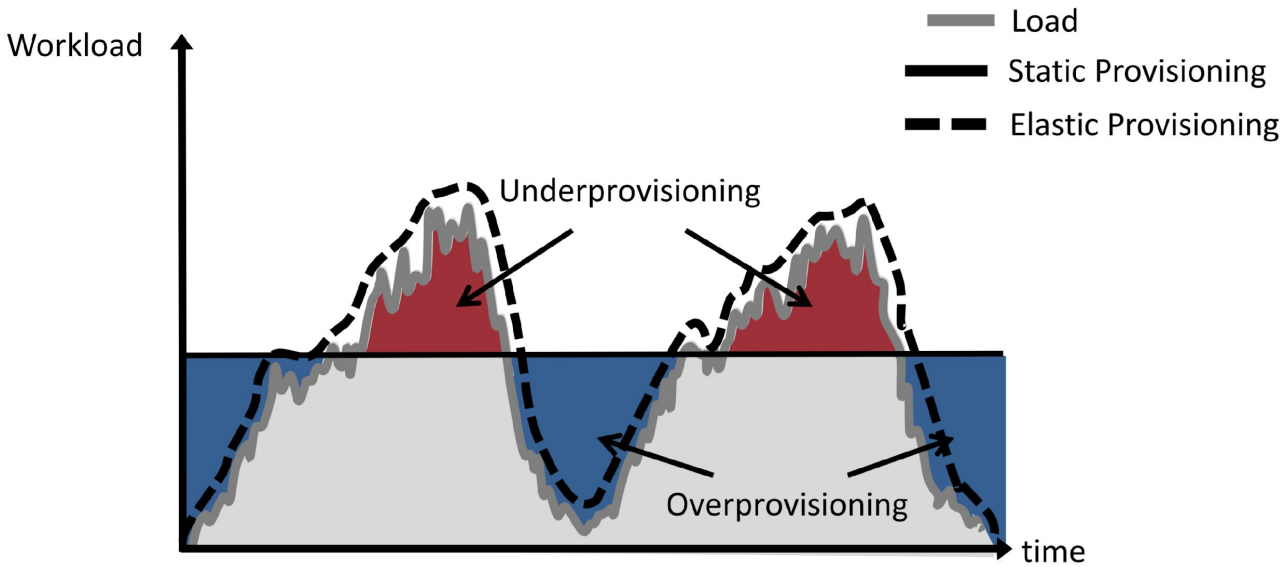


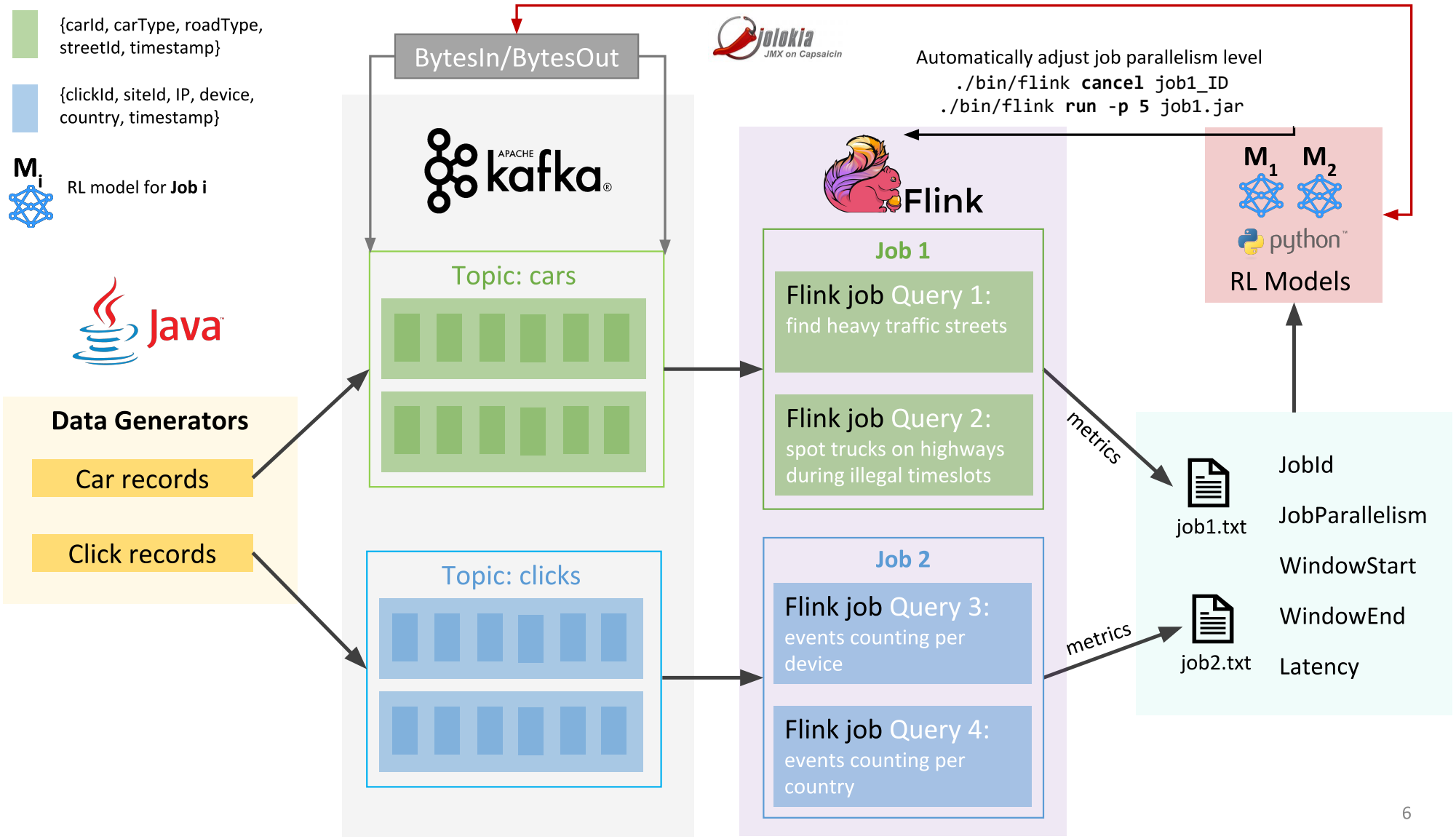
Image taken from [4]

Solution

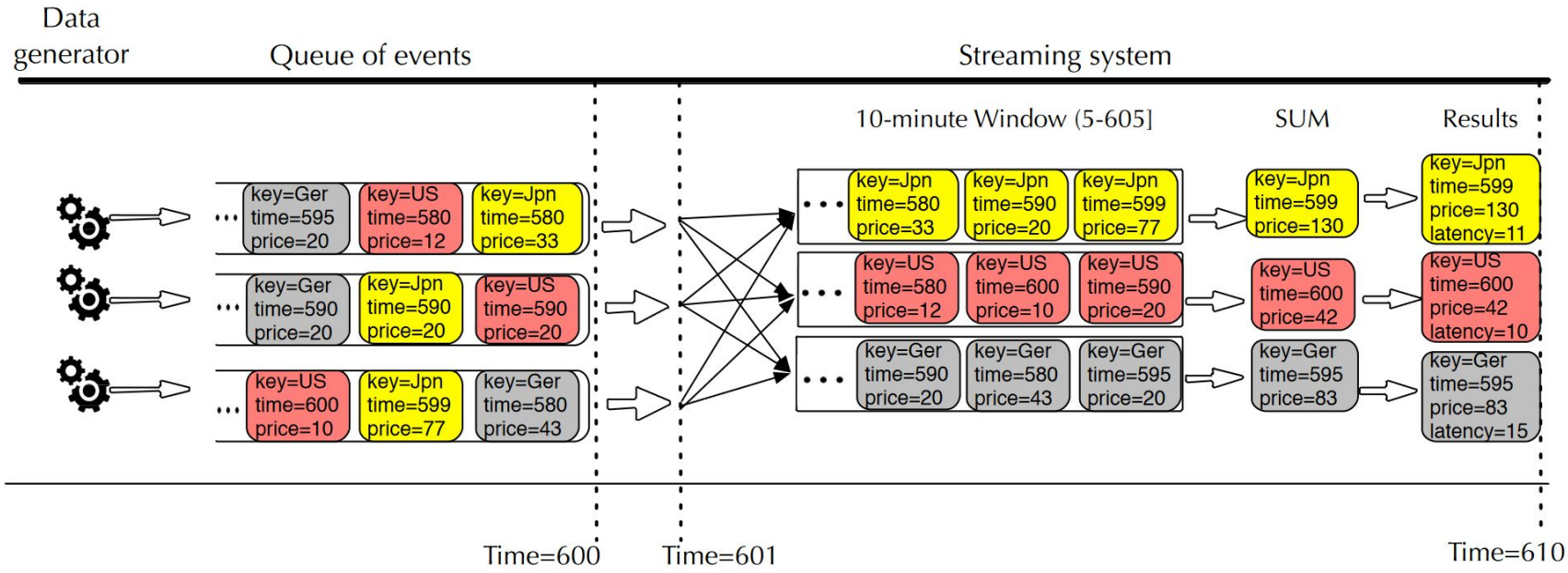
Machine learning model to monitor job execution and adjust its parallelism level, reacting to fluctuations on data generation rate

Machine learning everywhere. Taken from https://makeameme.org

RL


{carId, carType, roadType, streetId, timestamp}
{clickId, siteId, IP, device, country, timestamp}
M_i
RL model for **Job i**
Java
Data Generators
Car records
Click records
BytesIn/BytesOut
jolokia
JMX on Capsaicin
Automatically adjust job parallelism level
./bin/flink cancel job1_ID
./bin/flink run -p 5 job1.jar
APACHE kafka
Flink
M_1 M_2
python
RL Models
Topic: cars
Topic: clicks
Job 1
Flink job Query 1:
find heavy traffic streets
Flink job Query 2:
spot trucks on highways during illegal timeslots
Job 2
Flink job Query 3:
events counting per device
Flink job Query 4:
events counting per country
metrics
metrics
job1.txt
job2.txt
JobId
JobParallelism
WindowStart
WindowEnd
Latency

Computing Event-time Latency



[1] Karimov, Jeyhun, et al. "Benchmarking distributed stream data processing systems." *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018.

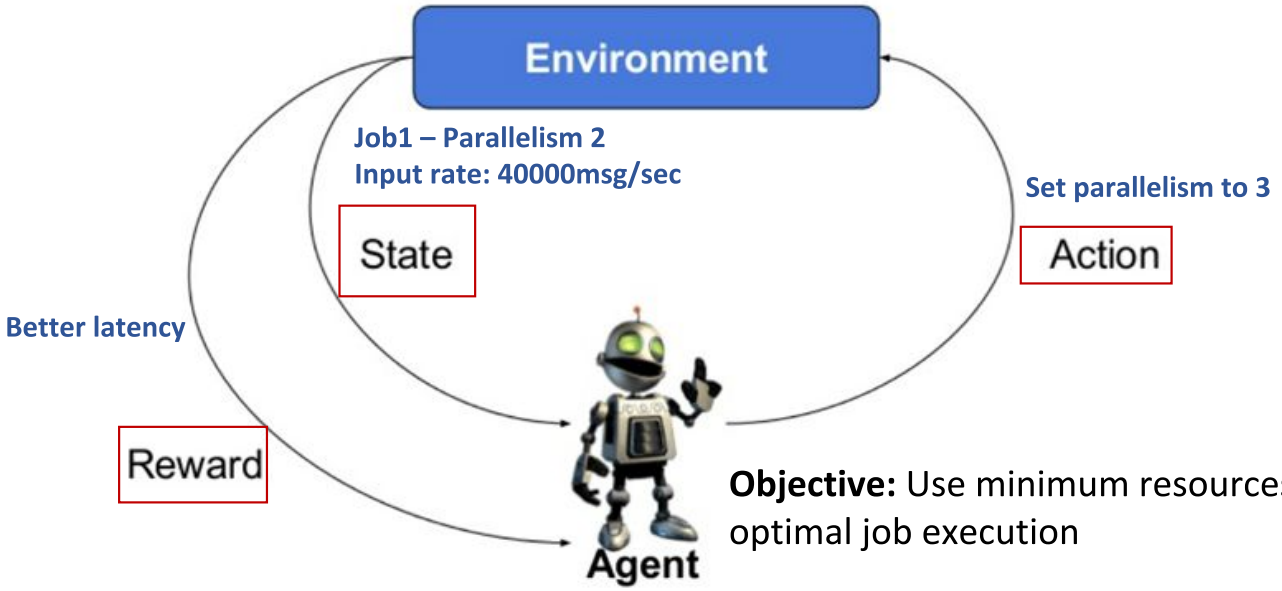
SASO Properties for control systems [9]

- *Stability* (not oscillating between different parallelism levels)
- *Accuracy* (finding the optimal parallelism for the given workload)
- *Settling time* (short time to reach the optimal parallelism)
- *Overshooting* (do not use more parallelism than optimal)

Reinforcement Learning Model

States, Actions, Reward, Algorithm, Optimizations

Idea



Objective: Use minimum resources to achieve optimal job execution

Reinforcement learning idea explained.
Image source: https://www.guru99.com/reinforcement-learning-tutorial.html

Reinforcement Learning – States & Actions

- State is defined by the **input rate**. We discretize the input rate for all the states we want to test
- States: **20 Bins** [7.5k, 15k], …, [142.5k, 150k] msg/sec
- Actions: **Parallelism** levels
- Max_parallelism = 10 (Kafka partitions = 10)

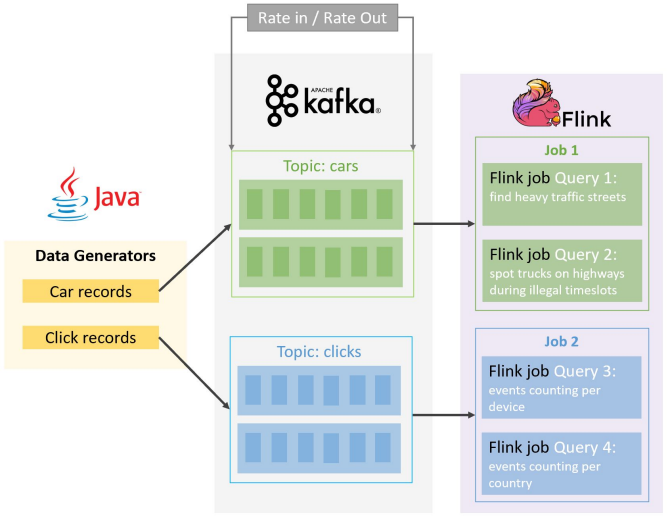
Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]										
[7500, 15000]										
[15000, 22500]										
[22500, 30000]										
[30000, 37500]										
[37500, 45000]										
[45000, 52500]										
[52500, 60000]										
[60000, 67500]										
[67500 ,75000]										
[75000 ,82500]										
[82500 ,90000]										
[90000 ,97500]										
[97500 ,105000]										
[105000 ,112500]										
[112500 ,120000]										
[120000 ,127500]										
[127500 ,135000]										
[135000 ,142500]										
[142500 ,150000]										

Reinforcement Learning – Reward

We measure input/output rate of *Kafka topic*

$$ratio_{io} = \frac{rate_{input}}{rate_{output}}$$

- $ratio_{io} \approx 1 \rightarrow good\ operation$
- $ratio_{io} > 1 \rightarrow backpressure\ eventually$
- $ratio_{io} < 1 \rightarrow not\ observable\ in\ our\ training$



We penalize higher parallelism with a coefficient **α** *(experiments: 0.3)*
The greater the value of α, the more penalty to higher parallelism levels

$$Reward = -ratio_{io} - \alpha \frac{parallelism}{parallelism_{max}}$$

Q- Learning Algorithm - Training

Algorithm 3: Simple RL Model

Data: states (input rate bins), actions (parallelism levels)
Result: Q-Table with values for optimal actions per state

1 $Q = Q_0$ (all cells to zero)
2 **for (** *each state in states* **)** {
3 Generate records at state's upper limit rate
4 **for (** *each action in actions* **)** {
5 Start a streaming job with *action* as parallelism
6 Sleep x minutes (allow execution to stabilize)
7 Measure current $ratio_io$
8 Calculate action reward r_a
9 # Update Q-Table with r_a
10 $Q[s, a] = r_a$
11 return Q

Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]										
[7500, 15000]										
[15000, 22500]										
[22500, 30000]										
[30000, 37500]										
[37500, 45000]										
[45000, 52500]										
[52500, 60000]										
[60000, 67500]										
[67500 ,75000]										
[75000 ,82500]										
[82500 ,90000]										
[90000 ,97500]										
[97500 ,105000]										
[105000 ,112500]										
[112500 ,120000]										
[120000 ,127500]										
[127500 ,135000]										
[135000 ,142500]										
[142500 ,150000]										

Training RL Model – Optimizations

$$best\ expected\ reward(p_2) = -1\ -0.2\ *\alpha = -1.06$$

- For a state:
  - if p1 < p2 and r1 > r2 then we don't need to try greater parallelism levels,
    - given that a desirable ratio_io is satisfied (experiments : 1.15)
  - if p1 < p2 and r2 cannot be greater than r1, skip greater parallelism levels
  - begin by testing the optimal parallelism of previous state

Cars Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]	-1.02	0	0	0	0	0	0	0	0	0
[7500, 15000]	-1.03	0	0	0	0	0	0	0	0	0
[15000, 22500]	-1.04	0	0	0	0	0	0	0	0	0
[22500, 30000]	-1.14	-1.06	0	0	0	0	0	0	0	0
[30000, 37500]	0	-1.07	0	0	0	0	0	0	0	0
[37500, 45000]	0	-1.06	0	0	0	0	0	0	0	0
[45000, 52500]	0	-1.08	0	0	0	0	0	0	0	0
[52500, 60000]	0	-1.16	-1.09	0	0	0	0	0	0	0
[60000, 67500]	0	0	-1.16	-1.12	0	0	0	0	0	0
[67500 ,75000]	0	0	0	-1.13	0	0	0	0	0	0
[75000 ,82500]	0	0	0	-1.12	0	0	0	0	0	0
[82500 ,90000]	0	0	0	-1.17	-1.15	0	0	0	0	0
[90000 ,97500]	0	0	0	0	-1.16	0	0	0	0	0
[97500 ,105000]	0	0	0	0	-1.16	0	0	0	0	0
[105000 ,112500]	0	0	0	0	-1.15	0	0	0	0	0
[112500 ,120000]	0	0	0	0	-1.14	0	0	0	0	0
[120000 ,127500]	0	0	0	0	-1.23	-1.28	0	0	0	0
[127500 ,135000]	0	0	0	0	-1.12	0	0	0	0	0
[135000 ,142500]	0	0	0	0	-1.37	-1.37	-1.37	-1.37	-1.33	-1.3
[142500 ,150000]	0	0	0	0	0	0	0	0	0	-1.31

$$Reward\ =\ -ratio_{io} - \alpha\ \frac{parallelism}{parallelism_{max}}$$

Testing RL Model - Algorithm

Algorithm 4: Simple RL Testing

Data: Running Job
Result: Running job with optimal parallelism

```
1 while True do
2     probe input rate to find state s
3     # Find optimal parallelism for s
4     opt = argmax_{Q≠0} Q(s)
5     set job parallelism to opt value
```

Algorithm 5: Adaptive RL Testing

Data: Running Job
Result: Running job with optimal parallelism, updated Q-Table

```
1 while True do
2     probe input rate to find state s
3     # Find optimal parallelism for s
4     opt = argmax_{Q≠0} Q(s)
5     set job parallelism to opt value
6     Measure current ratio_io and calculate action reward r_a
7     # Update Q-Table with r_a
8     Q[s, a] = r_a
```

Testing RL Model - (Changes for stability)

- **Scaling interval** - experiments: 30 seconds
  - Allow an interval of *scaling_interval* seconds between checking state
- **Stability period** - experiments: 60 seconds
  - If you are to change parallelism, allow for the state to stabilize for *stability_period* time
- **Rescale triggering latency (RTL)** - experiments : 2500ms
  - If latency >= RTL, select optimal parallelism from Q-Table according to current state
  - If latency < RTL, check (Q-Table) whether we need to decrease parallelism according to current state.

Experiments

Training clicks, cars jobs

Testing clicks, cars jobs on RL, Hand-tuned and Static models

Training RL Model

- IBM Cluster
- Data Generation:
  - 7.5k/sec up to 150k/sec
- States:
  - 20 Bins [0, 7.5k] … [142.5k, 150k] msg/sec
- Actions:
  - Max_parallelism: 10 (Kafka partitions: 10)
- Model performs an action for 5 mins on each state
- Exploring all space would take
  - 20 * 10 * 5 = 1000 min = **16.6 hours**
  - O(S * A) , S= # states, A=# actions

Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]										
[7500, 15000]										
[15000, 22500]										
[22500, 30000]										
[30000, 37500]										
[37500, 45000]										
[45000, 52500]										
[52500, 60000]										
[60000, 67500]										
[67500 ,75000]										
[75000 ,82500]										
[82500 ,90000]										
[90000 ,97500]										
[97500 ,105000]										
[105000 ,112500]										
[112500 ,120000]										
[120000 ,127500]										
[127500 ,135000]										
[135000 ,142500]										
[142500 ,150000]										

Cars Q-Table

Dropping from O(S*A) to **O(S+A)**

Converged in **2 hours**, which is one order of magnitude better

Cars Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]	-1.02	0	0	0	0	0	0	0	0	0
[7500, 15000]	-1.03	0	0	0	0	0	0	0	0	0
[15000, 22500]	-1.04	0	0	0	0	0	0	0	0	0
[22500, 30000]	-1.14	-1.06	0	0	0	0	0	0	0	0
[30000, 37500]	0	-1.07	0	0	0	0	0	0	0	0
[37500, 45000]	0	-1.06	0	0	0	0	0	0	0	0
[45000, 52500]	0	-1.08	0	0	0	0	0	0	0	0
[52500, 60000]	0	-1.16	-1.09	0	0	0	0	0	0	0
[60000, 67500]	0	0	-1.16	-1.12	0	0	0	0	0	0
[67500 ,75000]	0	0	0	-1.13	0	0	0	0	0	0
[75000 ,82500]	0	0	0	-1.12	0	0	0	0	0	0
[82500 ,90000]	0	0	0	-1.17	-1.15	0	0	0	0	0
[90000 ,97500]	0	0	0	0	-1.16	0	0	0	0	0
[97500 ,105000]	0	0	0	0	-1.16	0	0	0	0	0
[105000 ,112500]	0	0	0	0	-1.15	0	0	0	0	0
[112500 ,120000]	0	0	0	0	-1.14	0	0	0	0	0
[120000 ,127500]	0	0	0	0	-1.23	-1.28	0	0	0	0
[127500 ,135000]	0	0	0	0	-1.12	0	0	0	0	0
[135000 ,142500]	0	0	0	0	-1.37	-1.37	-1.37	-1.37	-1.33	-1.3
[142500 ,150000]	0	0	0	0	0	0	0	0	0	-1.31

Clicks Q-Table

Dropping from O(S*A) to O(S+A)

Converged in **3 hours**, which is one order of magnitude better

Clicks Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]	-1.03	0	0	0	0	0	0	0	0	0
[7500, 15000]	-1.03	0	0	0	0	0	0	0	0	0
[15000, 22500]	-1.04	0	0	0	0	0	0	0	0	0
[22500, 30000]	-1.03	0	0	0	0	0	0	0	0	0
[30000, 37500]	-1.3	-1.06	0	0	0	0	0	0	0	0
[37500, 45000]	0	-1.06	0	0	0	0	0	0	0	0
[45000, 52500]	0	-1.06	0	0	0	0	0	0	0	0
[52500, 60000]	0	-1.15	-1.09	0	0	0	0	0	0	0
[60000, 67500]	0	0	-1.1	0	0	0	0	0	0	0
[67500 ,75000]	0	0	-1.15	-1.12	0	0	0	0	0	0
[75000 ,82500]	0	0	0	-1.12	0	0	0	0	0	0
[82500 ,90000]	0	0	0	-1.19	-1.15	0	0	0	0	0
[90000 ,97500]	0	0	0	0	-1.15	0	0	0	0	0
[97500 ,105000]	0	0	0	0	-1.15	0	0	0	0	0
[105000 ,112500]	0	0	0	0	-1.15	0	0	0	0	0
[112500 ,120000]	0	0	0	0	-1.18	-1.36	0	0	0	0
[120000 ,127500]	0	0	0	0	-1.25	-1.48	0	0	0	0
[127500 ,135000]	0	0	0	0	-1.3	-1.52	0	0	0	0
[135000 ,142500]	0	0	0	0	-1.41	-1.6	-1.64	-1.67	-1.71	-1.42
[142500 ,150000]	0	0	0	0	-1.45	-1.69	-1.66	-1.7	-1.77	-1.48

Hand-tuned model

Our prediction on which is the best parallelism for each state, assuming **1.7** scalability, defined after numerous experiments

Parallelism of 1 consumes up to 30k per sec, so parallelism of 2 is expected to consume 30k * 1.7 = 51k per sec

Hand-tuned Qtable	Actions - Parallelism level									
States - bins	1	2	3	4	5	6	7	8	9	10
[0, 7500]	-1	0	0	0	0	0	0	0	0	0
[7500, 15000]	-1	0	0	0	0	0	0	0	0	0
[15000, 22500]	-1	0	0	0	0	0	0	0	0	0
[22500, 30000]	-1	0	0	0	0	0	0	0	0	0
[30000, 37500]	0	-1	0	0	0	0	0	0	0	0
[37500, 45000]	0	-1	0	0	0	0	0	0	0	0
[45000, 52500]	0	-1	0	0	0	0	0	0	0	0
[52500, 60000]	0	0	-1	0	0	0	0	0	0	0
[60000, 67500]	0	0	-1	0	0	0	0	0	0	0
[67500 ,75000]	0	0	-1	0	0	0	0	0	0	0
[75000 ,82500]	0	0	-1	0	0	0	0	0	0	0
[82500 ,90000]	0	0	-1	0	0	0	0	0	0	0
[90000 ,97500]	0	0	0	-1	0	0	0	0	0	0
[97500 ,105000]	0	0	0	-1	0	0	0	0	0	0
[105000 ,112500]	0	0	0	-1	0	0	0	0	0	0
[112500 ,120000]	0	0	0	-1	0	0	0	0	0	0
[120000 ,127500]	0	0	0	-1	0	0	0	0	0	0
[127500 ,135000]	0	0	0	-1	0	0	0	0	0	0
[135000 ,142500]	0	0	0	-1	0	0	0	0	0	0
[142500 ,150000]	0	0	0	0	-1	0	0	0	0	0

Experiment results for Cars & Clicks jobs

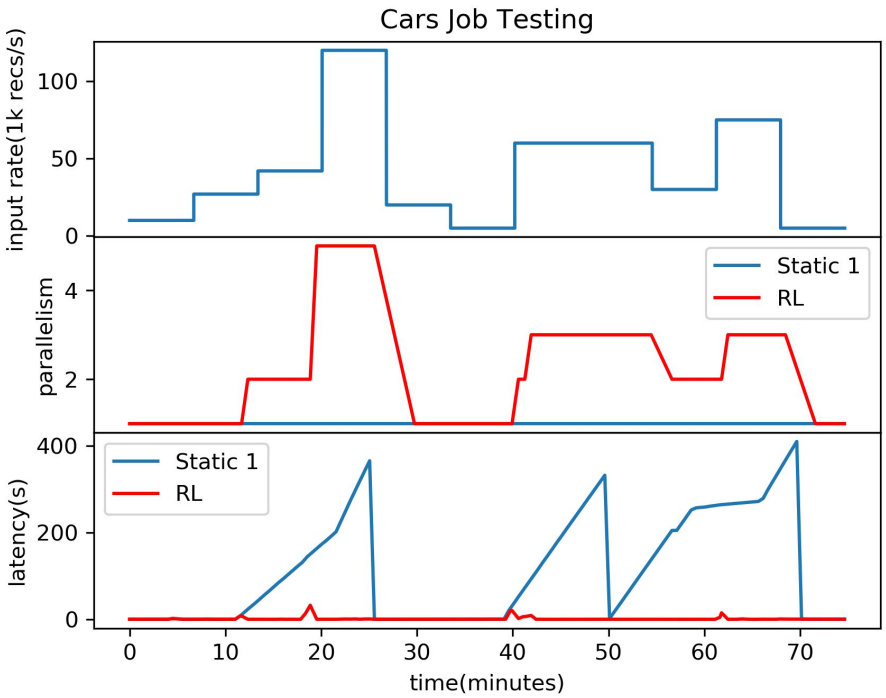
- We tested the following models based on the following data generation pattern ->
  - Reinforcement Learning trained model
  - Hand-tuned model
  - Static Parallelism of 1
  - Static Parallelism of 5
  - Static Parallelism of 10

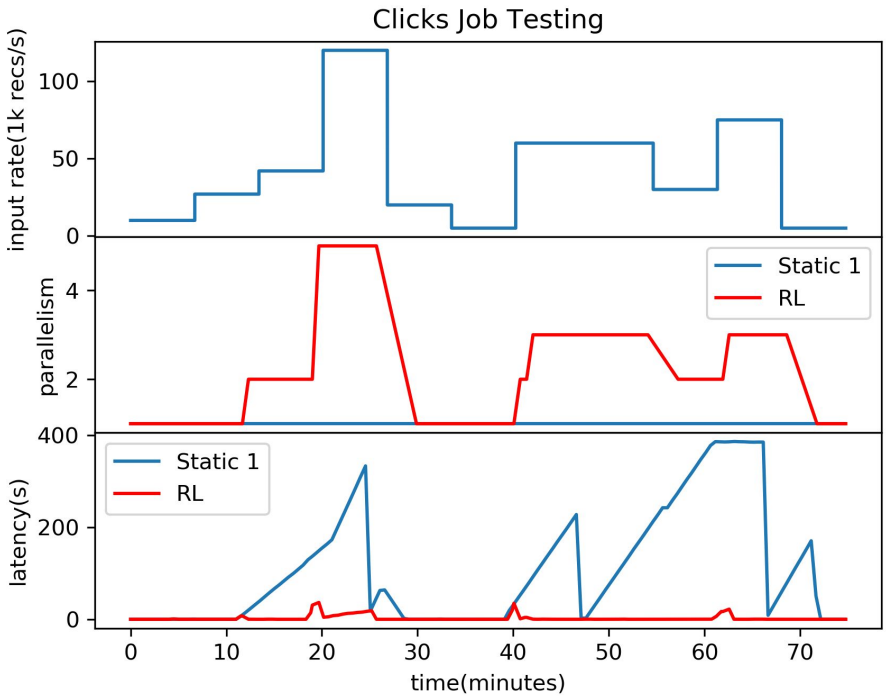
- Each experiment runs for **78 mins**

msg/sec, mins

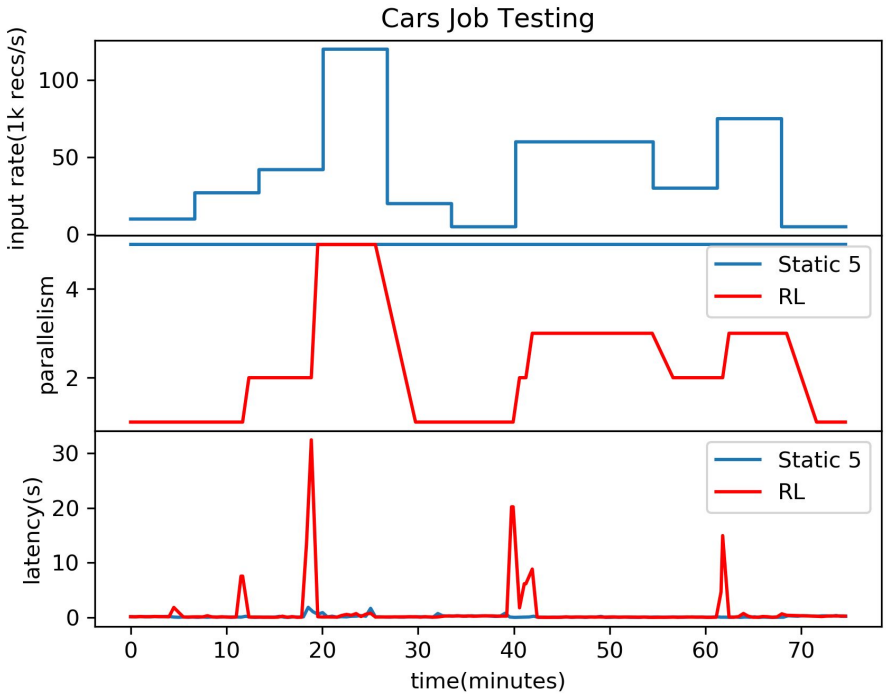
```
[(10000, 7),
(27000, 7),
(42000, 7),
(120000, 7),
(20000, 7),
(5000, 7),
(60000, 15),
(30000, 7),
(75000, 7),
(5000, 7)]
```

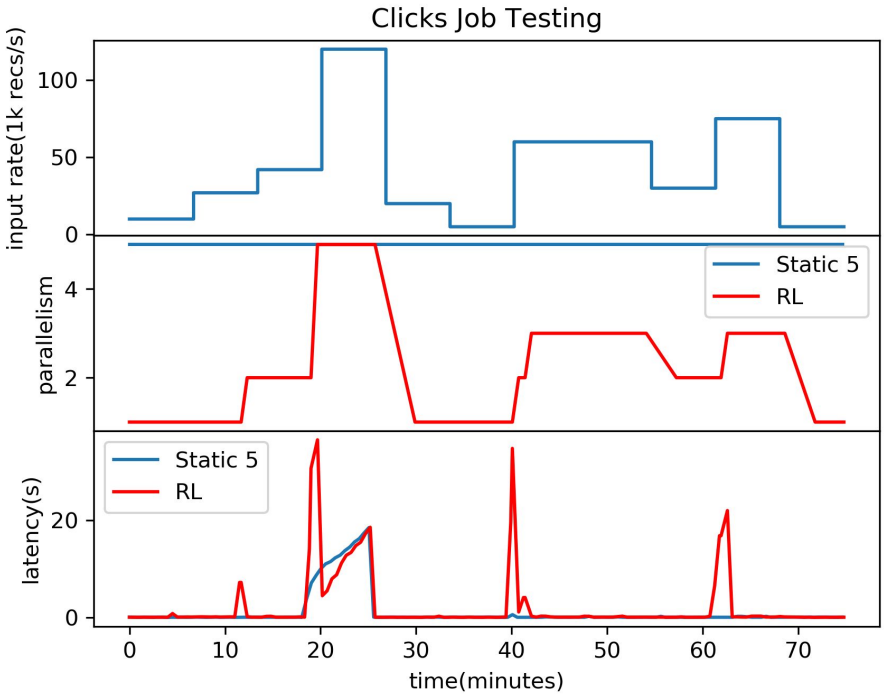
Experiment results: Static 1



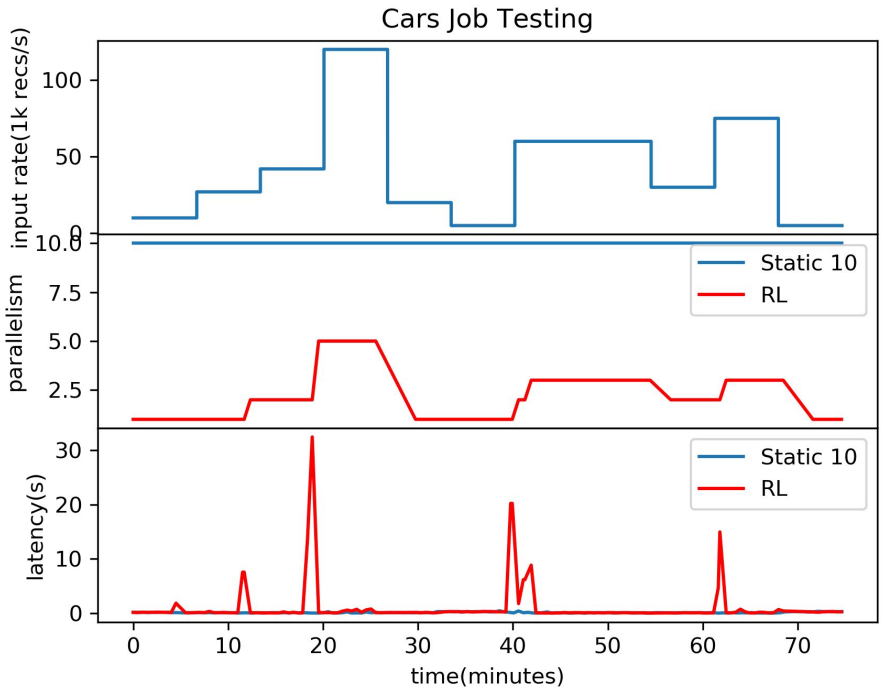


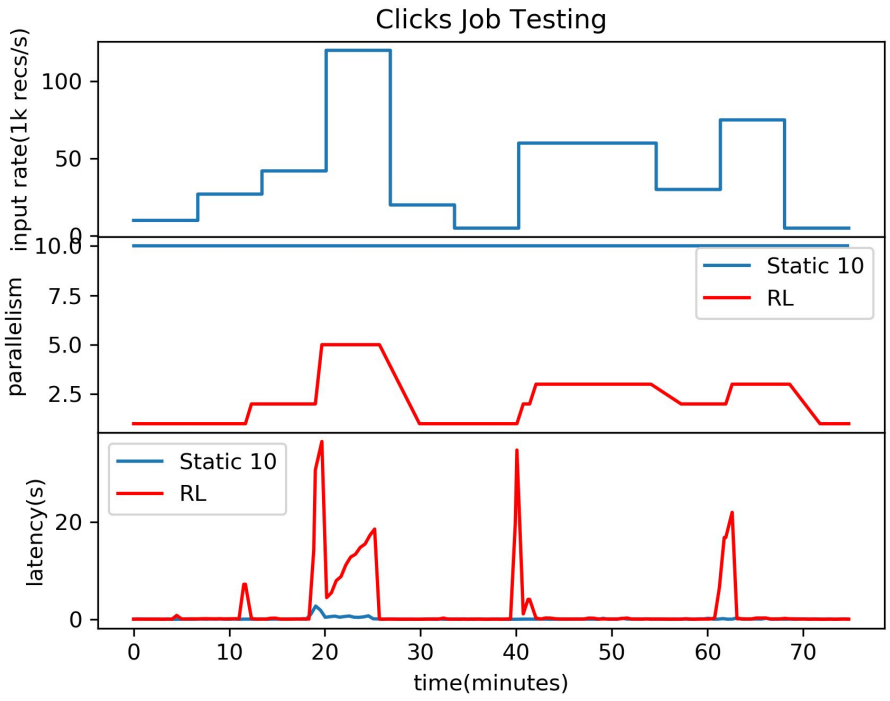
Experiment results: Static 5



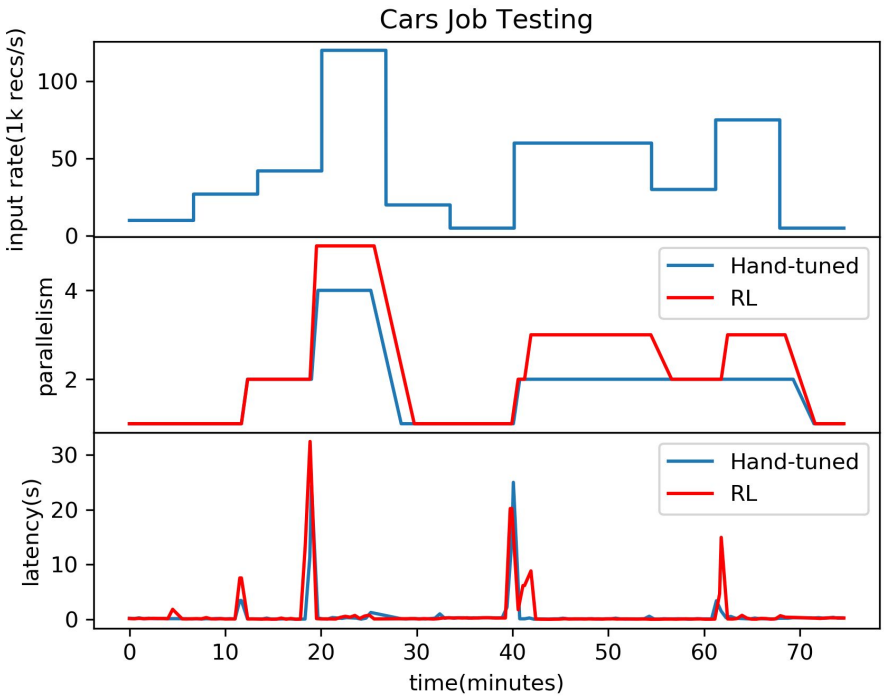


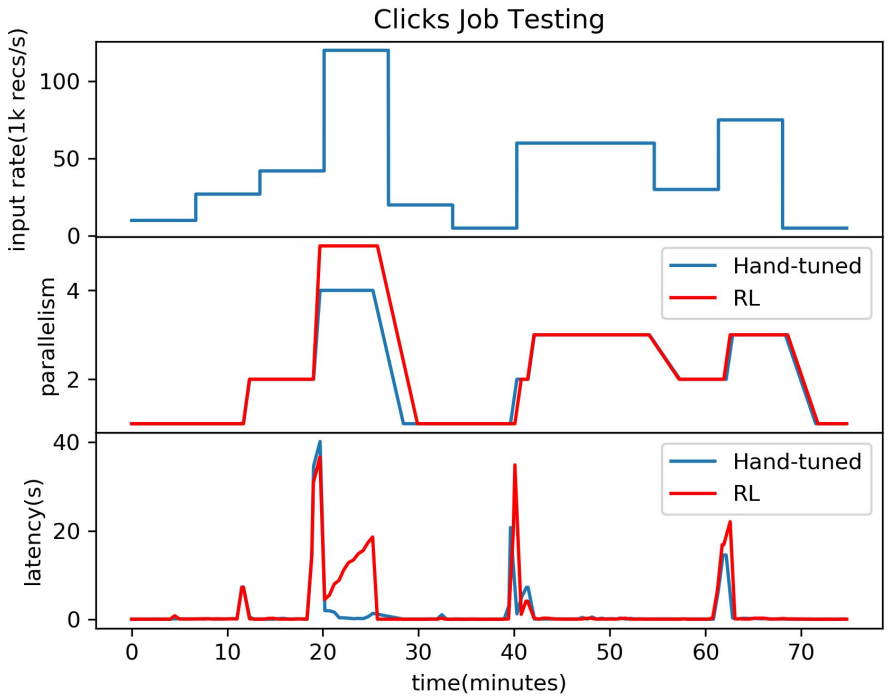
Experiment results: Static 10





Experiment results: Hand-tuned





Conclusion

- Static provision is a poor fit for long-running streaming applications
- We propose an auto-scaling controller for distributed streaming dataflows, leveraging Reinforcement Learning with Q-Learning
- Our reactive model selects the optimal action in one step, being adaptive to any kind of workload and environment

Future Work

- Keep state – Start from savepoint when restarting Flink job
- Experiment with adaptivity
- An extension of our RL model for controlling operator-level parallelism scaling, by collecting metrics from the processing engine (e.g. Flink Metrics) [8] and decoupling from the messaging system
- Model-based reinforcement learning to train for multiple jobs, not needing different training for different jobs.

References

[1] Karimov, Jeyhun, et al. "Benchmarking distributed stream data processing systems." *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 2018.

[2] Apache Flink documentation:

- Parallel execution: https://ci.apache.org/projects/flink/flink-docs-stable/dev/parallel.html
- Metrics API: https://ci.apache.org/projects/flink/flink-docs-stable/monitoring/metrics.html
- Monitor Back-pressure: https://ci.apache.org/projects/flink/flink-docs-stable/monitoring/back_pressure.html

[3] Lohrmann, Björn, Peter Janacik, and Odej Kao. "Elastic stream processing with latency guarantees." 2015 IEEE 35th International Conference on Distributed Computing Systems. IEEE, 2015.

- Presentation slides: https://www.slideshare.net/bjoernlohrmann/onsite-presentation-28372119

[4] Heinze, Thomas, et al. "Auto-scaling techniques for elastic data stream processing." 2014 IEEE 30th International Conference on Data Engineering Workshops. IEEE, 2014.

- Presentation slides: https://pdfs.semanticscholar.org/33af/8c7137af6314f1d4fd420dfcfd0a74e22092.pdf

[5] Intro to Reinforcement Learning: https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/

[6] Gedik, Buğra, et al. "Elastic scaling for data stream processing." IEEE Transactions on Parallel and Distributed Systems 25.6 (2013): 1447-1463.

[7] https://en.wikipedia.org/wiki/Q-learning

[8] https://flink.apache.org/2019/07/23/flink-network-stack-2.html

[9] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. 2004.Feedback control of computing systems. John Wiley & Sons