SBAC-PAD 2019

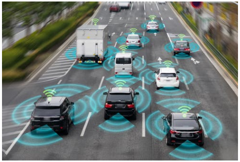
INVENTORS FOR THE DIGITAL WORLD

Monte-Carlo Tree Search and Reinforcement Learning for Reconfiguring Data Stream Processing on Edge Computing

Alexandre Veith, Marcos Dias de Assunção, Laurent Lefèvre

University of Lyon, ENS of Lyon, Claude Bernard University Lyon 1
CNRS, Inria, Parallel Computing Lab (LIP), Lyon, FRANCE

Data Stream Processing Scenarios



- Application scenarios[1]
  - Monitoring of operational infrastructure and precision agriculture
  - Anomaly detection, fraud detection
  - Smart cities, smart homes, traffic control, autonomous vehicles
  - Wearable assistance, augmented reality
- Applications generate unbounded streams of data
- Data stream processing in the Cloud
  - **Multiple tiers of data collection and processing**
  - Data in motion systems, message brokers, that increase latency
- Edge computing for data stream processing

[1] Pictures are a courtesy of Google images

Cloud and Edge Computing

Cloud

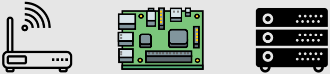
Internet

- Data storage
- Batch and stream processing
- Data warehousing
- Business applications

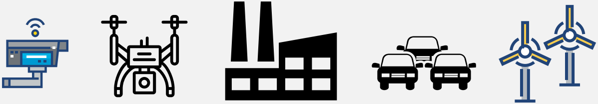
Edge

LAN/WAN

- Real-time data processing
- Basic analytics
- Data filtering, optimisation
- Data caching, buffering

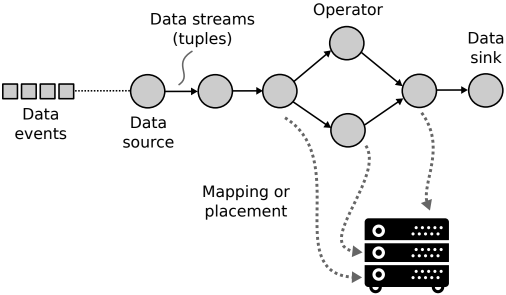


Sensors and Controlers



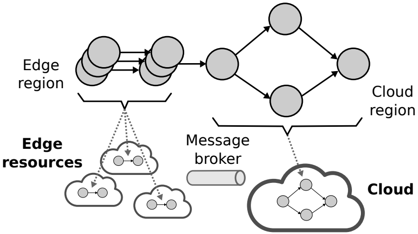
Data Stream Processing Dataflows[1]

- Applications are structured as directed graphs
- Operator properties
  - Selectivity
  - Data transformation
  - State
- Operators are assigned to resources (**placement**)



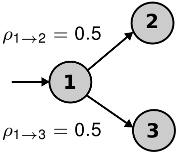
[1] M. D. Assunção *et al.*, Resource Elasticity for Distributed Data Stream Processing: A Survey and Future Directions, Journal of Network and Computer Applications, Vol. 103, pp. 1-17, Feb. 2018.

Modelling the Placement across Cloud and Edge

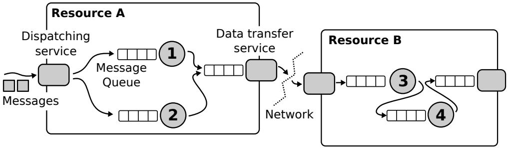


- **Infrastructure** graph $\mathcal{N} = (\mathcal{R}, \mathcal{L})$ of compute resources $\mathcal{R}$ and logical links $\mathcal{L}$
  - Resources have CPU and memory capabilities
  - Network links have bandwidth and latency

- **Application** DAG $\mathcal{G} = (\mathcal{O}, \mathcal{E})$ of operators $\mathcal{O}$ and streams $\mathcal{E}$, where an operator's requirements comprise:
  - CPU MIPS to process an event
  - Memory to load the operator
  - Selectivity
  - Data transformation
- Probability $\rho_{i \to j}$ that an output event emitted by operator *i* will flow through to operator *j*



Modelling the Placement across Cloud and Edge – Cont.



- Operators and communication services handle events in a FCFS basis
- Both services are modelled as M/M/1 queues

- L_{p_i}: end-to-end latency of path p_i is the sum of the computation time of all operators in p_i and the communication time to stream events along p_i
- **Placement goal**: find a mapping $\mathcal{M} : \mathcal{O} \rightarrow \mathcal{R}, \mathcal{E} \rightarrow \mathcal{L}$ that minimises the Aggregate end-to-end Latency (AL) of all paths:

$$AL = \min \sum_{p_i \in \mathcal{P}} L_{p_i}$$

where $\mathcal{P}$ is the set of all paths in the application DAG[a]

[a] A. Veith *et al.*, Latency-Aware Placement of Data Stream Analytics on Edge Computing, ICSOC 2018, pp. 215-229, Hangzhou, China, Nov. 2018.

Need for Application Reconfiguration and its Goal

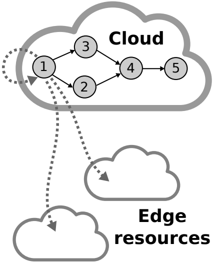
- Data stream processing applications are long-running
- Workload conditions may change over time
- Initial placement might not be ideal
- Resources at the edge are more failure prone

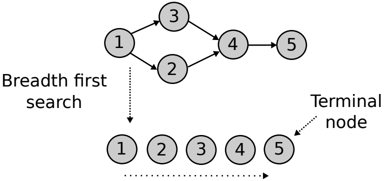
Reconfiguration goal: Find a new mapping $\mathcal{M} : \mathcal{O} \rightarrow \mathcal{R}, \mathcal{E} \rightarrow \mathcal{L}$ that improves the current Aggregate end-to-end Latency

Markov Decision Process (MDP) and Reconfiguration

- **MDP** comprises a set of states $\mathcal{S}$, where each state $s \in \mathcal{S}$ has a number of possible actions $\mathcal{A}(s)$ and a reward function $R(s)$
  - State s contains a mapping $\mathcal{M} : \mathcal{O} \rightarrow \mathcal{R}, \mathcal{E} \rightarrow \mathcal{L}$
  - Action $a \in \mathcal{A}(s)$ is either migrating an operator to another resource or maintaining its current mapping
  - The reward $R(s)$ reflects how much the aggregate end-to-end latency is improved under state s:

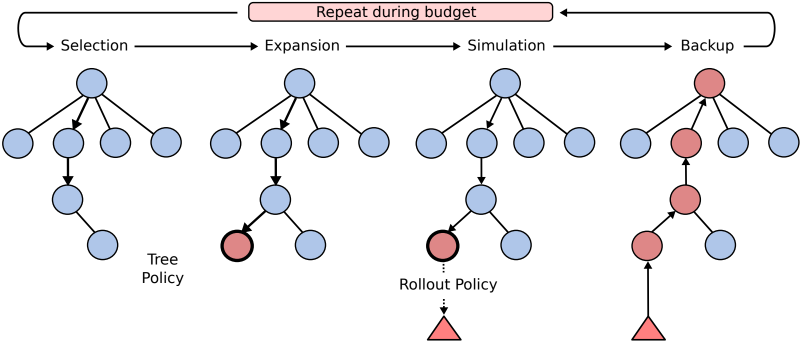
$$R(s) = AL_{s_0} - AL_s$$





- **An episode** is a set of transitions from an initial state to a terminal state
- **An optimal policy** defines the transitions from states to actions that maximise the reward

Monte-Carlo Tree Search[1]



- In addition to a valid placement, a node/state *s* contains:
  - A count $N(s)$ with number of times the state was visited
  - An action value $Q(s, a)$ for each action
  - A count $N(s, a)$ of times an action *a* was picked
- Simulated episode is created using *tree policy* and *rollout policy*
  - Exploration versus exploitation dilemma
- Generated return is used to update/initialise the action values

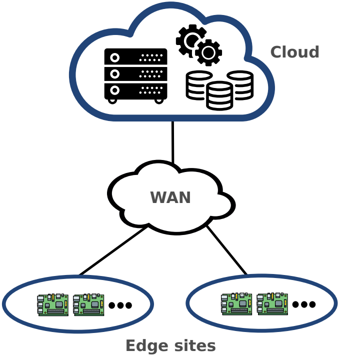
[1] R. S. Sutton and A. G. Barto, Reinforcement Learning: An introduction. MIT press, 2018.

MCTS-Best-UCT and Deployment Hierarchy (DH)

- MCTS-UCT:
  - It assigns a bonus to the uncertainty in the value of a state-action
  - Its tree policy picks the action that maximises the Upper Confidence Bound (UCB)
- **MCTS-Best-UCT**:
  - It maintains a list of visited nodes with their UCB values
  - Instead of starting the tree search from the root node, its "tree policy" picks the node with the best UCT value from the list
- Deployment Hierarchy:
  - Action space can be large as the number of resources grows
  - Operators on a path with a sink on the edge have priority
  - DH sorts operators by their potential impact on end-to-end latency[1]

[1] A. Veith *et al.*, Latency-Aware Placement of Data Stream Analytics on Edge Computing, ICSOC 2018, pp. 215-229, Hangzhou, China, Nov. 2018.

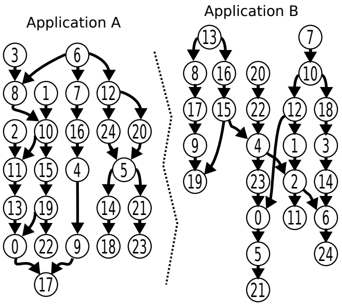
Experimental Setup



- Discrete-event simulation (OMNET++)
- One cloud with 2 servers and two edge sites with 20 resources each
  - Cloud servers are modelled as AMD Ryzen 7 1800x
  - Edge servers as Raspberry Pi's model 2
- Edge resources are interconnected by a LAN whereas the communication among sites is done via a WAN (Internet)
- Network latency is modelled based on experiments conducted in previous work[1]

[1] W. Hu *et al.*, Quantifying the impact of edge computing on mobile applications, in 7th ACM SIGOPS Asia-Pacific Workshop on Systems, pp. 5:1–5:8, New York, USA 2016.

Evaluated Applications



- The number of operators is based on the graph order of RIoTBench[1] applications
- For each application, 15 different configurations were created by varying the following operator properties:

Operator property	Value
cpu	1–100 MIPS
Data transf. ratio	10–100%
mem	100–7,500 Bytes
Input event size	100–2500 Bytes
Selectivity	10–100%
Input event rate	1,000–10,000 messages

- The initial placement of sources and sinks changes in each configuration
- The sink on the critical path is always placed on the cloud

[1] https://github.com/dream-lab/riot-bench

Performance Evaluation Scenarios

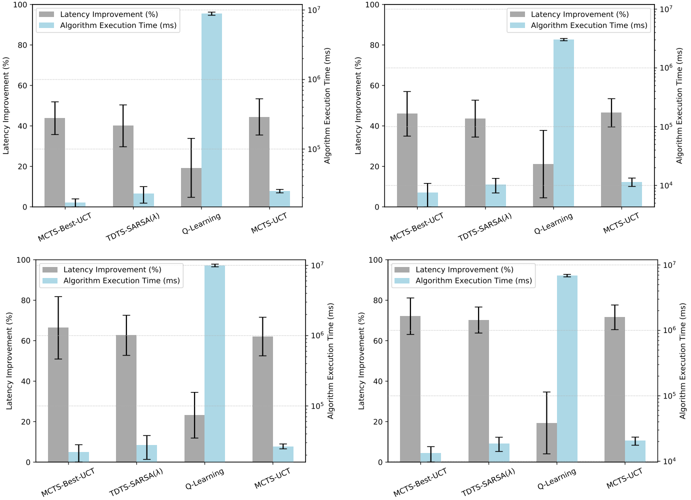
- **Scenario 1:** Reinforcement algorithms receive a cloud-only deployment as initial placement (all operators placed in the cloud)
  - Q-learning, TDTS-Sarsa(λ)
  - With and without Deployment Hierarchy
- **Scenario 2:** Evaluating the aggregate end-to-end latency, it considers all reinforcement learning algorithms and previously proposed solutions
  - Taneja's algorithm, RTR and RTR-RP

- Execution budget is 10,000 iterations/episodes
- Initial placement is run for 300 seconds or until all application paths have processed at least 500 messages each

Performance Metrics

- Latency improvement
- Algorithm execution time
- Time to best latency
- Number of operator migrations
- Minimum aggregate end-to-end latency

Latency Improvement



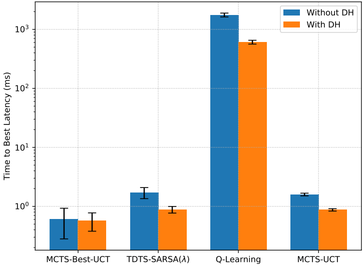
Application A

Application B

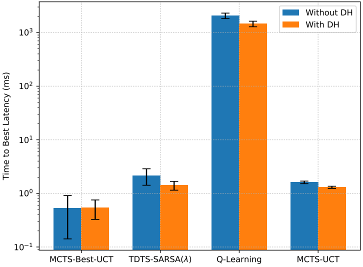
(a) Without DH

(b) With DH

Time to Achieve the Best Latency



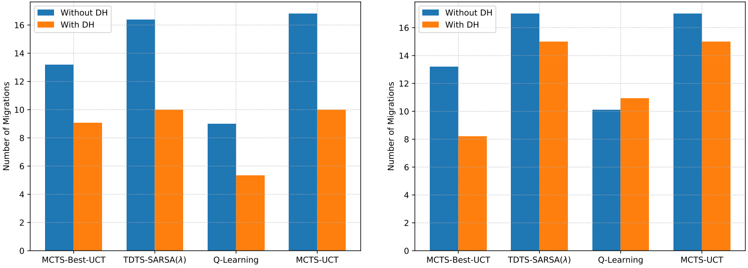
(a) Application A



(b) Application B

- **Application A:** MCTS-Best-UCT performs at least 64% better that MCTS-UCT without DH and 33% with DH
- **Application B:** MCTS-Best-UCT also performs best

Number of Operator Migrations

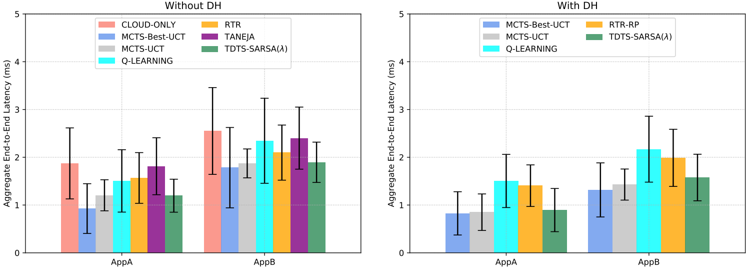


(a) Application A

(b) Application B

- MCTS-Best-UCT discovers earlier on the operators that have the biggest impact on latency (*i.e.*, operators that are selective) and migrates them to edge resources

Minimum Aggregate End-to-End Latency



- The reinforcement learning algorithms improve the latency compared to other solutions from the state of the art
- Expect for MCTS-Best-UCT and Q-learning, the solutions proposed by the reinforcement learning algorithms are more stable

Conclusions and Future Work

- Summary and conclusions:
  - Markov Decision Process for DSP application reconfiguration
  - Evaluation of reinforcement learning algorithms
  - MCTS-Best-UCT improves the *time to best latency*
  - MCTS-Best-UCT is also able to achieve *end-to-end latency* similar to other algorithms under a smaller budget

- Future work:
  - Evaluate the algorithms on a real testbed
  - Use other machine learning techniques to approximate the Q-values (deep reinforcement learning)
  - Use energy consumption as an optimisation metric

Questions?

assuncao@acm.org