Quality of Service Aware Mechanisms for (Re)Configuring Data Stream Processing Applications on Highly Distributed Infrastructure

Alexandre DA SILVA VEITH

alexandre.veith@ens-lyon.fr

Supervisors:

Laurent LEFEVRE - INRIA

Marcos DIAS DE ASSUNCAO - INRIA

23rd September 2019

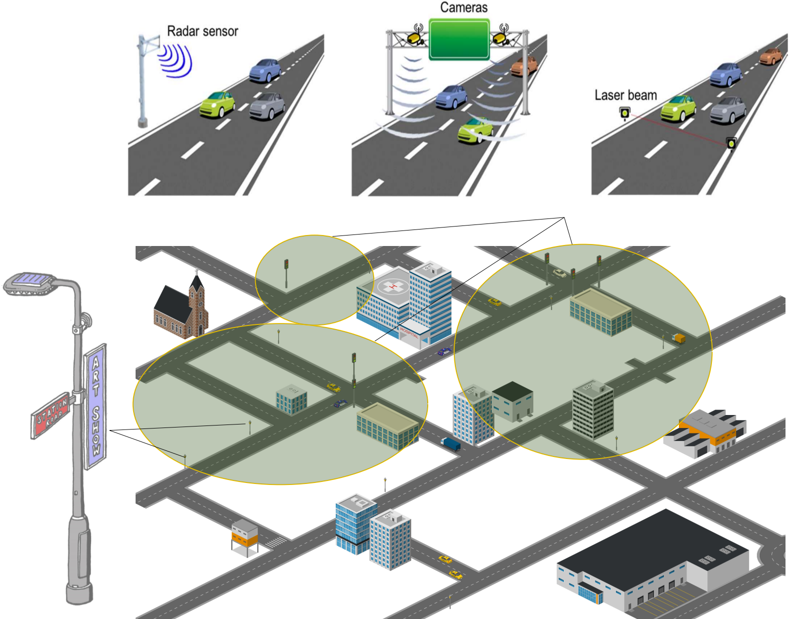
Motivation

Wearable Assistance

Traffic Control

- Multiple sensors collect data continuously.
- Data must be analysed, as it is generated.

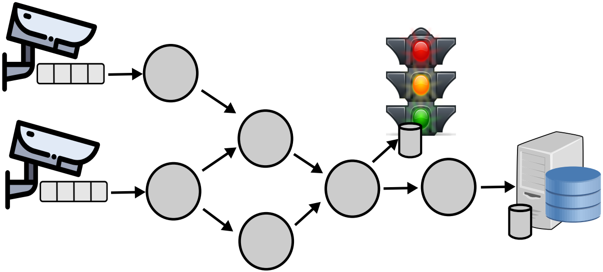
Use Case - Smart Traffic Light Management



https://eu-smartcities.eu/initiatives/78/description

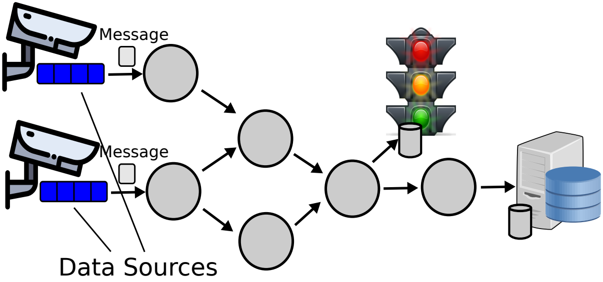
Data Stream Processing (DSP) Engines

Data Stream Processing Engines supply the environment for processing data in a timely manner and under most engines the application is structured as a **dataflow**.



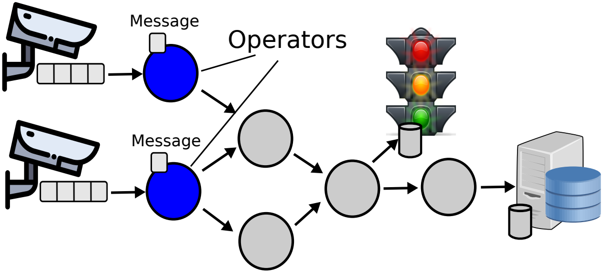
Data Stream Processing Engines supply the environment for processing data in a timely manner and under most engines the application is structured as a **dataflow**.

- Data sources.



Data Stream Processing Engines supply the environment for processing data in a timely manner and under most engines the application is structured as a **dataflow**.

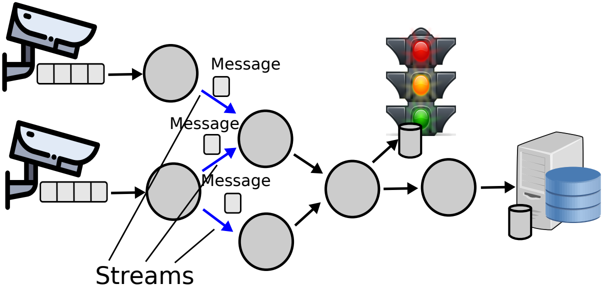
- Data sources.
- Operators.



Data Stream Processing (DSP) Engines

Data Stream Processing Engines supply the environment for processing data in a timely manner and under most engines the application is structured as a **dataflow**.

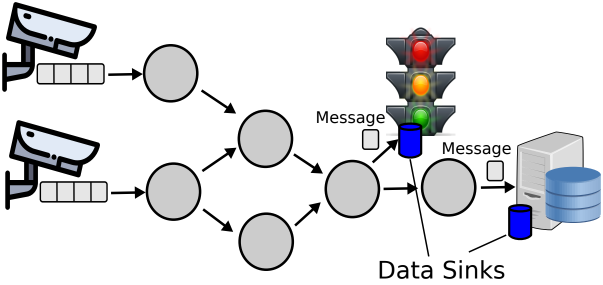
- Data sources.
- Operators.
- Streams.



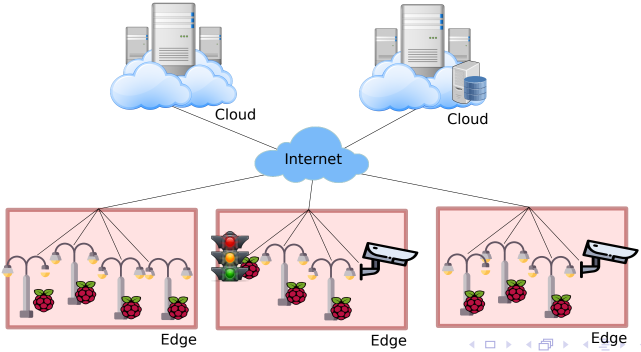
Data Stream Processing (DSP) Engines

Data Stream Processing Engines supply the environment for processing data in a timely manner and under most engines the application is structured as a **dataflow**.

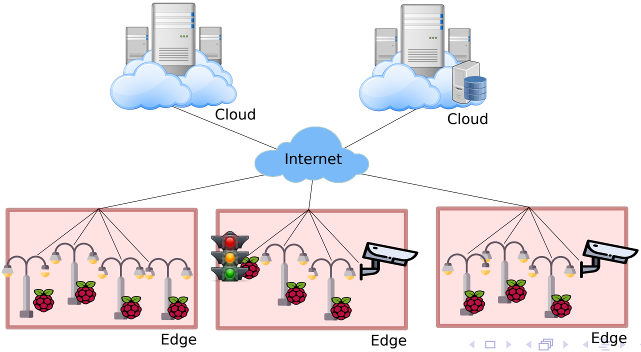
- Data sources.
- Operators.
- Streams.
- Data sinks.



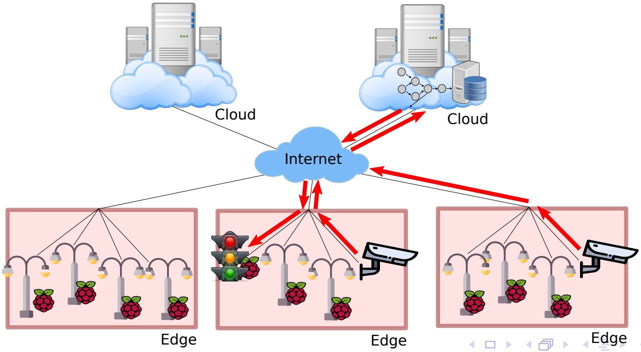
(Re)Configuration of DSP Applications



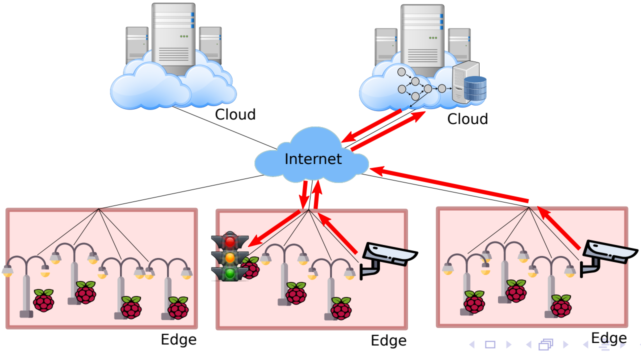
- **Initial operator placement (application configuration).**



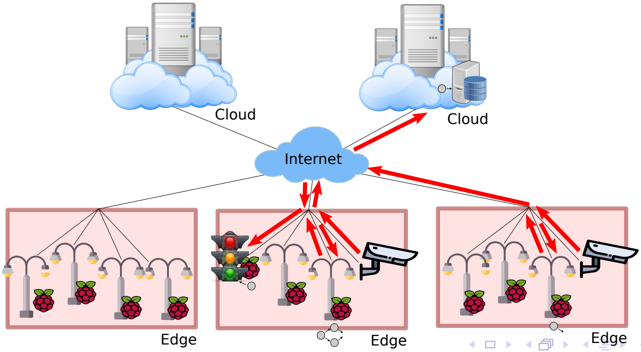
- **Initial operator placement (application configuration)**.
  - Application placement on the cloud (traditional).



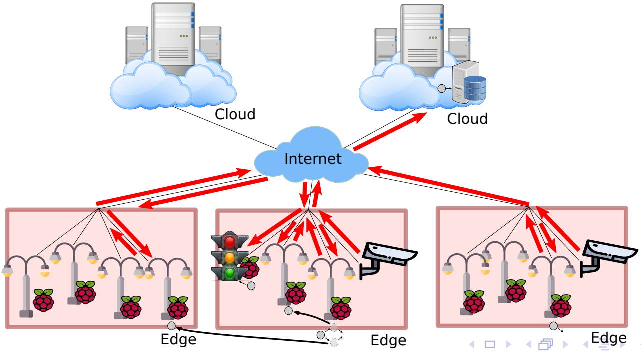
- **Initial operator placement (application configuration)**.
  - Application placement on the cloud (traditional).
    - **High communication overhead - hard to achieve (near) real-time data analytics (Hu et al., 2016)**.



- **Initial operator placement (application configuration)**.
  - Application placement on the cloud (traditional).
    - **High communication overhead - hard to achieve (near) real-time data analytics (Hu et al., 2016)**.
  - Application placement on edge resources.
    - Limited CPU, memory and bandwidth capabilities.



- **Initial operator placement (application configuration)**.
  - Application placement on the cloud (traditional).
    - **High communication overhead - hard to achieve (near) real-time data analytics (Hu et al., 2016)**.
  - Application placement on edge resources.
    - Limited CPU, memory and bandwidth capabilities.
- **Application reconfiguration**.
  - Workload changes, infrastructure changes, ...
  - Problem more complex than the application configuration.



Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).
- Infrastructure graph:
  - Heterogeneous computing capabilities (cpu and memory).
  - Heterogeneous network links (bandwidth and latency).

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).
- Infrastructure graph:
  - Heterogeneous computing capabilities (cpu and memory).
  - Heterogeneous network links (bandwidth and latency).

Challenges:

Research Problem and Challenges

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).
- Infrastructure graph:
  - Heterogeneous computing capabilities (cpu and memory).
  - Heterogeneous network links (bandwidth and latency).

Challenges:

- How to handle **heterogeneous computing resources**?

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).
- Infrastructure graph:
  - Heterogeneous computing capabilities (cpu and memory).
  - Heterogeneous network links (bandwidth and latency).

Challenges:

- How to handle **heterogeneous computing resources**?
- How to use edge resources considering their **computing and communication limitations**?

Research Problem and Challenges

Research Problem:

How to (re)configure time-sensitive DSP applications efficiently across heterogeneous edge and cloud resources?

Problem Details:

- Application graph:
  - Heterogeneous operator requirements (memory and cpu) and properties (selectivity, data transformation, stateless or stateful).
  - Heterogeneous stream requirements (bandwidth).
- Infrastructure graph:
  - Heterogeneous computing capabilities (cpu and memory).
  - Heterogeneous network links (bandwidth and latency).

Challenges:

- How to handle **heterogeneous computing resources**?
- How to use edge resources considering their **computing and communication limitations**?
- How to overcome the **communication overhead** when sending messages via the Internet?

Literature Review* and Positioning

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).

- Communication overhead is neglected (Taneja and Davy, 2017; Foroni et al., 2018).

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

Literature Review* and Positioning

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).

- Communication overhead is neglected (Taneja and Davy, 2017; Foroni et al., 2018).

- Previous works focus on data sinks located in the cloud (Ni et al., 2017; Mortazavi et al., 2017).

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

Literature Review* and Positioning

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).
    - Edge and cloud heterogeneous computing resources.

- Communication overhead is neglected (Taneja and Davy, 2017; Foroni et al., 2018).

- Previous works focus on data sinks located in the cloud (Ni et al., 2017; Mortazavi et al., 2017).

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

Literature Review* and Positioning

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).
  - Edge and cloud heterogeneous computing resources.
- Communication overhead is neglected (Taneja and Davy, 2017; Foroni et al., 2018).
  - High latency of the Internet links must be considered.
- Previous works focus on data sinks located in the cloud (Ni et al., 2017; Mortazavi et al., 2017).

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

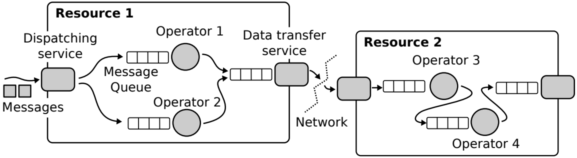
Literature Review* and Positioning

- Homogeneous computing resources (Apache Heron, Apache Storm, Apache Spark).
  - Edge and cloud heterogeneous computing resources.
- Communication overhead is neglected (Taneja and Davy, 2017; Foroni et al., 2018).
  - High latency of the Internet links must be considered.
- Previous works focus on data sinks located in the cloud (Ni et al., 2017; Mortazavi et al., 2017).
  - Data sinks placed both on the cloud and on edge resources.

* Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

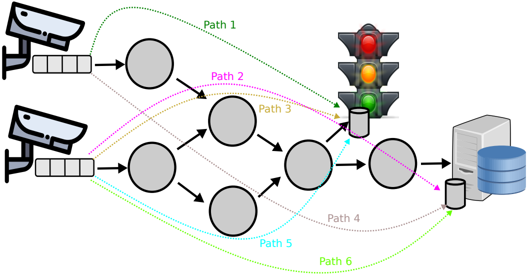
Table of Contents

Deployment and System Overview

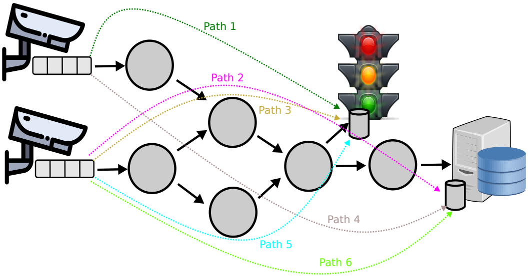


- Queues for computation (operator) and communication (data transfer service).
- Each message queue handles messages in a First-Come, First-Served (FCFS) fashion for preserving messages time order.
- Model is based on M/M/1 queueing theory model. Arrivals follow a Poisson distribution (constant rate) and the service rate follows an exponential distribution (homogeneous processing time).
- Heterogeneous operator properties (selectivity, data transformation, stateless or stateful).

Aggregate End-to-End Latency

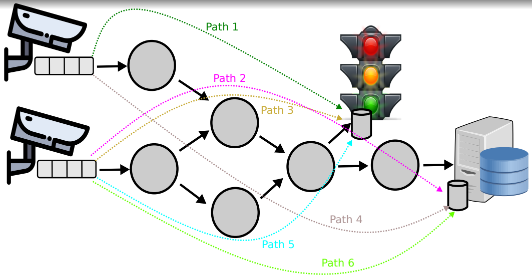


End-to-end latency: time for messages to traverse a path from a data source to a data sink.



End-to-end latency: time for messages to traverse a path from a data source to a data sink.

$$\textbf{Aggregate End-to-End Latency} = \sum_{p_i \in \text{paths}} \text{end-to-end latency of } p_i$$



End-to-end latency: time for messages to traverse a path from a data source to a data sink.

$$\textbf{Aggregate End-to-End Latency} = \sum_{p_i \in \text{paths}} \text{end-to-end latency of } p_i$$

Contribution

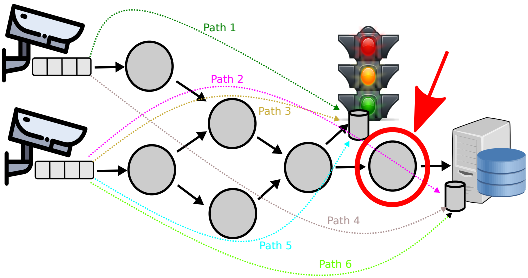
Two application configuration strategies (RTR and RTR+RP) for minimising the **Aggregate End-to-End Latency**.

Response Time Rate - RTR

```
list ← Breadth-First Search (BFS) Traversal algorithm (Peng et al., 2015);
min_cost ← ∞;
elected ← null;
deployment ← { };
for each operator at list do
    for each resource at available resources do
        total_cost, constraints ← simulate the aggregate end-to-end latency using model;
        if total_cost is shorter than min_cost and !constraints then
            min_cost ← total_cost;
            elected ← evaluated resource;
        end
    end
    deployment ← deployment ∪ elected;
end
return deployment;
```

RTR Issues

- RTR leads to a **combinatorial explosion** when evaluating possibilities resulting into a high computational cost.
- There is **waste of computing capabilities on edge resources** due to the multiple application path requirements.



RTR with Region Patterns - RTR+RP

```
list ← BFS Traversal algorithm (Peng et al., 2015);
min_cost ← ∞;
elected ← null;
deployment ← { };
for each operator at list do
    for each resource at region(operator) do
        total_cost, constraints ← simulate the aggregate end-to-end
          latency using model;
        if total_cost is shorter than min_cost and !constraints then
            min_cost ← total_cost;
            elected ← evaluated resource;
        end
    end
    if elected is equal to null then
        deployment ← deployment ∪ cloud resource;
    else
        deployment ← deployment ∪ elected;
    end
end
return deployment;
```

Two sets of experiments:

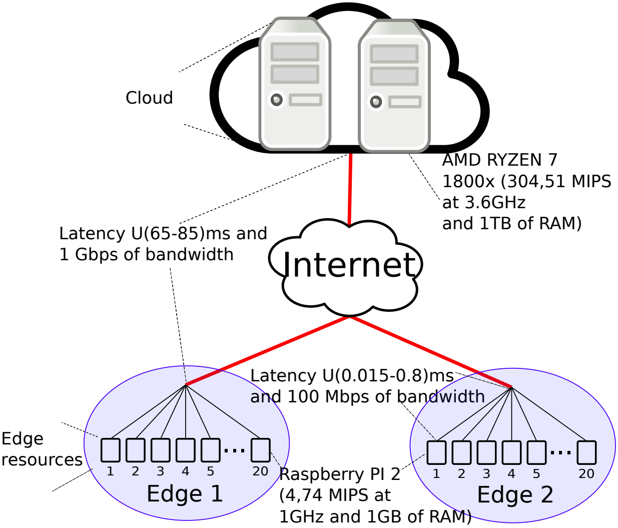
- **Simulations**.
- A **real testbed**.

Simulation - Performance Comparison

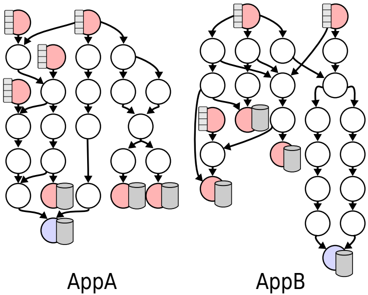
- State-of-the-art:
  - Traditional approach (**Cloud**).
  - **LB** (Taneja and Davy, 2017), which considers CPU, memory, and bandwidth constraints to obtain the operator placement.

Simulation - Experimental Setup

- Developed atop **OMNET++**.
- Network topology based on (Hu et al., 2016).



Simulation - Evaluated Applications



Labels

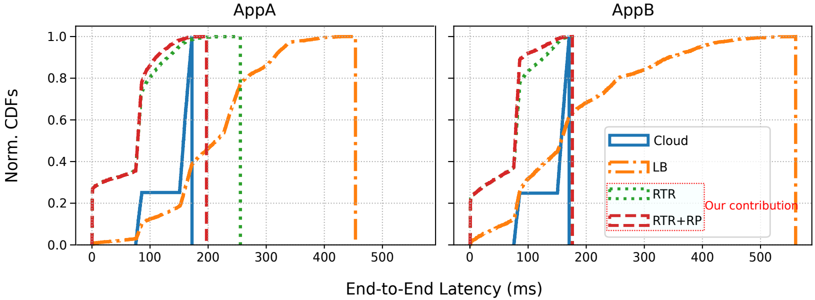
- Edge Infrastructure
- Cloud Infrastructure
- Data source
- Sink

Operator Parameters

Parameter	Value	Unit
cpu	1000-10000	Instructions per second
Data transformation rate	0-90	%
mem	100-7500	bytes
Input message size	100-2500	bytes
Selectivity	0-90	%
Input message rate	1000-10000	Messages/second

- Each application dataflow results in **300** new dataflows.
- Each experiment runs during **60 seconds** of simulation time.

Simulation - Results on End-to-End Latency



- Our contributions outperform in over **50%** and **57%** the Cloud and the LB, respectively.

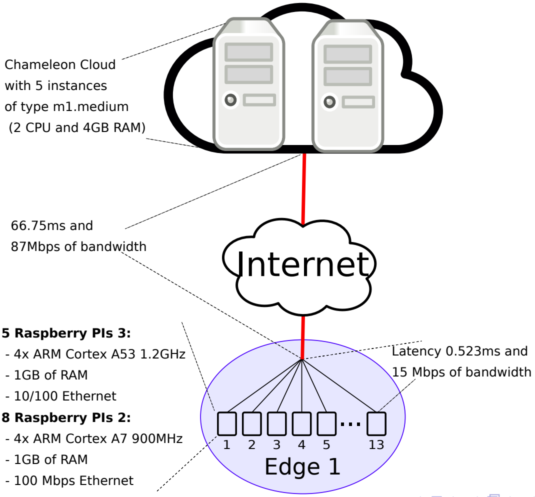
Alexandre da Silva Veith and Marcos Dias de Assunção and Laurent Lefèvre. Latency-Aware Placement of Data Stream Analytics on Edge Computing. ICSOC 2018.

Real Testbed - Comparison

- State-of-the-art:
  - Traditional approach (**Cloud**).
  - **LB** (Taneja and Davy, 2017), which considers CPU, memory, and bandwidth constraints to obtain the operator placement.
- **Random** which is the average of 15 different dataflow deployments across edge and cloud resources.

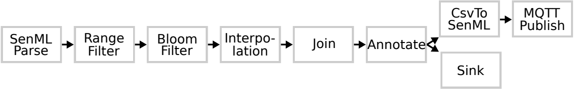
Real Testbed - Experimental Setup

- Implemented in **R-Pulsar** (rpulsar.org), which is a "All-in-one" **lightweight engine** for efficient and real-time data-driven stream processing.



Real Testbed - Evaluated Application

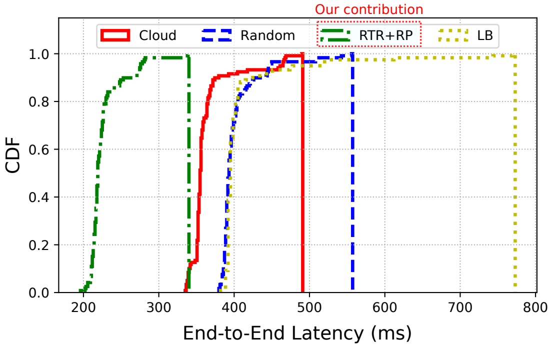
Extract, Transform and Load (ETL) application is part of **RIoTBench** (Shukla, Chaturvedi and Simmhan, 2017), which is an IoT benchmark for DSP systems.



The experiments were conducted using Sense Your City dataset (http://map.datacanvas.org) which consists of :

- Data collected from sensors spread across **7 cities** (an average of 12 sensor per city) in **3 continents**;
- Each message includes metadata with timestamped observations of **outdoor temperature, humidity, ambient light, dust, and air quality**.

Real Testbed - Results on End-to-End Latency



RTR+RP reduces in over **38%** the end-to-end latency compared to cloud and **44%** when compared to Random and LB.

Eduard Gibert Renart, and **Alexandre da Silva Veith** and Daniel Balouek-Thomert and Marcos Dias de Assunção and Laurent Lefèvre and Manish Parashar. Distributed Operator Placement for IoT Data Analytics Across Edge and Cloud Resources. CCGrid 2019.

Table of Contents

Application Reconfiguration Issues

- DSP applications are **long-running**.
- **Changes can happen** during the application execution (workload, infrastructure, performance requirements, ...).
- Changes lead to **reorganise** or **migrate** operators across available computing resources.
- **Large search space** for determining the application reconfiguration.

Application Reconfiguration

- Reinforcement Learning (RL) has demonstrated to be **efficient with large search space problems**:
  - Board games (Gelly and Silver, 2011).
  - DSP application placement (Russo et al., 2018).

Application Reconfiguration

- Reinforcement Learning (RL) has demonstrated to be **efficient with large search space problems**:
  - Board games (Gelly and Silver, 2011).
  - DSP application placement (Russo et al., 2018).
- RL algorithms **learn** by interacting with an environment.

- Reinforcement Learning (RL) has demonstrated to be **efficient with large search space problems**:
  - Board games (Gelly and Silver, 2011).
  - DSP application placement (Russo et al., 2018).
- RL algorithms **learn** by interacting with an environment.
- The environment is often a **Markov Decision Process (MDP)** and it models how the system behaves.

Application Reconfiguration

- Reinforcement Learning (RL) has demonstrated to be **efficient with large search space problems**:
  - Board games (Gelly and Silver, 2011).
  - DSP application placement (Russo et al., 2018).
- RL algorithms **learn** by interacting with an environment.
- The environment is often a **Markov Decision Process (MDP)** and it models how the system behaves.

Contribution

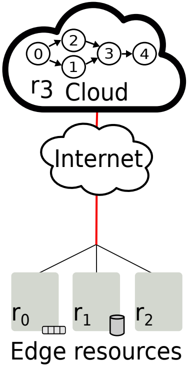
A Markov Decision Process (MDP) model and the evaluation of RL algorithms considering a multi-objective problem optimisation for the application reconfiguration.

Modelling the Reconfiguration as an MDP

Formally, an MDP comprises:

- State space
  - All possible mappings of operator/stream onto resource/link(s).

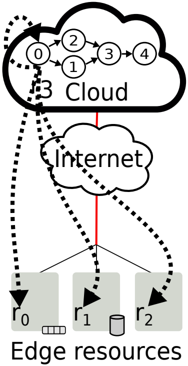
Current application deployment



Modelling the Reconfiguration as an MDP

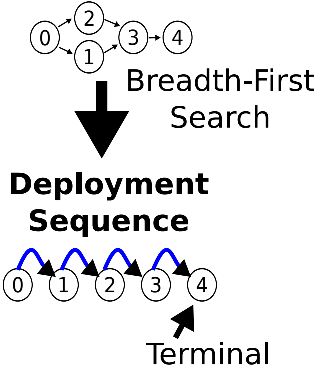
Formally, an MDP comprises:

- State space
  - All possible mappings of operator/stream onto resource/link(s).
- Action space
  - Each possible action consists in maintaining the current mapping of a given operator or migrating it to another resource.



Formally, an MDP comprises:

- State space
  - All possible mappings of operator/stream onto resource/link(s).
- Action space
  - Each possible action consists in maintaining the current mapping of a given operator or migrating it to another resource.
- Transition function
  - Transitions are given by the deployment sequence.



Modelling the Reconfiguration as an MDP

Formally, an MDP comprises:

- State space
  - All possible mappings of operator/stream onto resource/link(s).
- Action space
  - Each possible action consists in maintaining the current mapping of a given operator or migrating it to another resource.
- Transition function
  - Transitions are given by the deployment sequence.
- Reward function

Quality of Service (QoS) metrics:

- **End-to-end latency:** time for messages to traverse a path from a data source to a data sink.
- **WAN traffic**
  data volume crossing WAN network links.
- **Monetary cost**
  calculates the monetary cost using the number of connections and exchanged messages across the cloud and the edges, and vice-versa (*Azure IoT Hub* 2019; *AWS IoT Core* 2019).
- **Reconfiguration Overhead**
  The total downtime incurred by migrating operator code and state.

Modelling the Reconfiguration as an MDP

Formally, an MDP comprises:

- State space
  - All possible mappings of operator/stream onto resource/link(s).
- Action space
  - Each possible action consists in maintaining the current mapping of a given operator or migrating it to another resource.
- Transition function
  - Transitions are given by the deployment sequence.
- Reward function

Single aggregate cost $=$
$w_0 \times$ End-to-end latency
$+w_1 \times$ WAN traffic
$+w_2 \times$ Monetary cost
$+w_3 \times$ Reconfiguration Overhead

where w corresponds to weight assigned to the metric.

Aggregate cost $=$

$$\sum_{p_i \in \text{paths}} \text{single aggregate cost of } p_i$$

Modelling the Reconfiguration as an MDP

Formally, an MDP comprises:

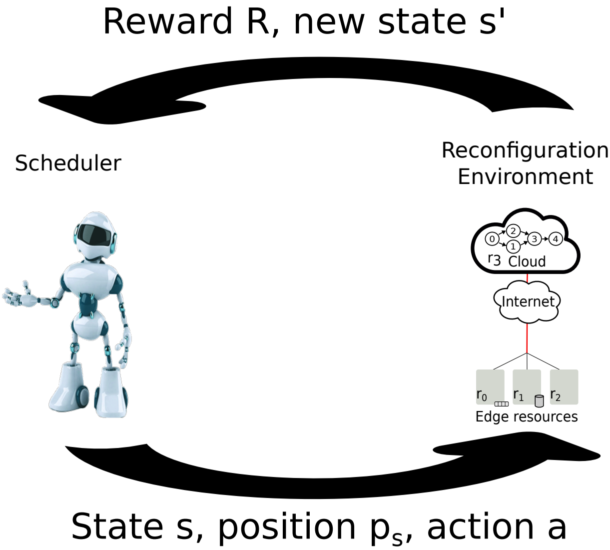
- State space
  - All possible mappings of operator/stream onto resource/link(s).
- Action space
  - Each possible action consists in maintaining the current mapping of a given operator or migrating it to another resource.
- Transition function
  - Transitions are given by the deployment sequence.
- Reward function
  - Reward =Aggregate cost (current deployment) - Aggregate cost (reconfiguration deployment)

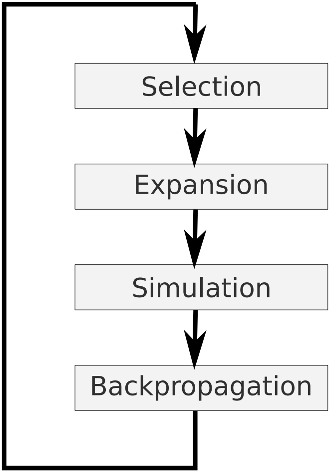
Single aggregate cost $=$
$w_0 \times$ End-to-end latency
$+w_1 \times$ WAN traffic
$+w_2 \times$ Monetary cost
$+w_3 \times$ Reconfiguration Overhead

where w corresponds to weight assigned to the metric.

$$\textbf{Aggregate cost} = \sum_{p_i \in \text{paths}} \text{single aggregate cost of } p_i$$

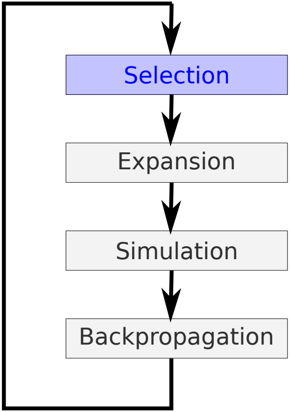
Solving the MDP Problem





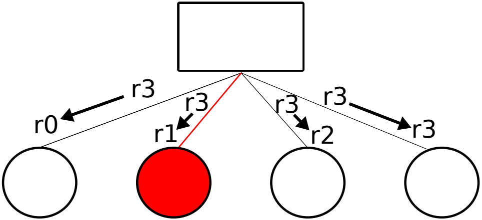
- A **budget** within a number of iterations is used to execute the MCTS loop.
- The algorithm **builds a decision-tree** with possible reconfiguration deployments.

Solving the MDP Problem with MCTS Algorithms

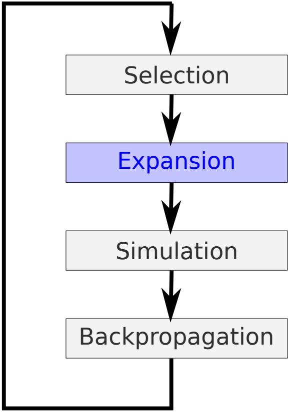



```
while node is not terminal do
    if node is fully expanded then
        Get the most promising node;
    else
        return node;
    end
end
```

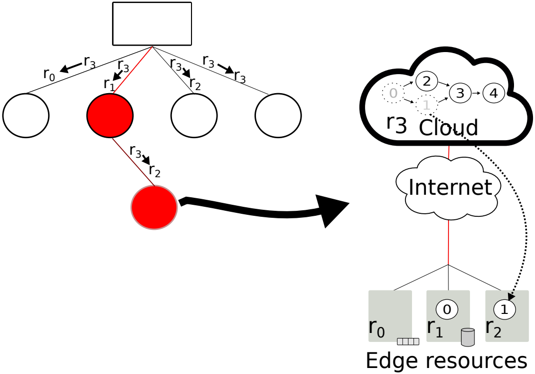
Upper Confidence Bound for Trees (UCT) algorithm (Sutton and Barto, 2018) provides the most promising node.



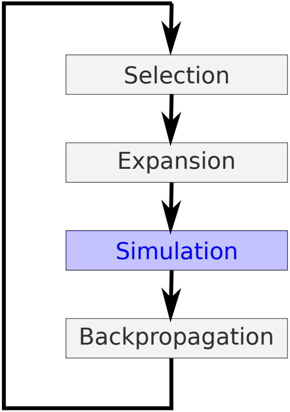
Solving the MDP Problem with MCTS Algorithms



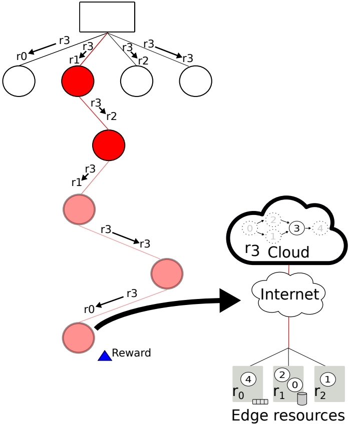
1 choose an untried action;
2 add a new child to the selected node;
3 return the new node;



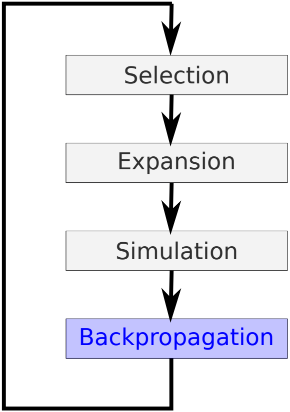
Solving the MDP Problem with MCTS Algorithms



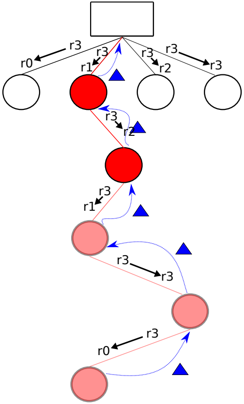
while *node is not terminal* **do**
    Choose an action randomly;
end
Simulate the new placement and determine its reward;



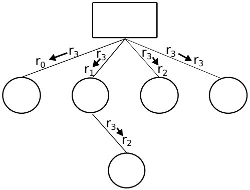
Solving the MDP Problem with MCTS Algorithms



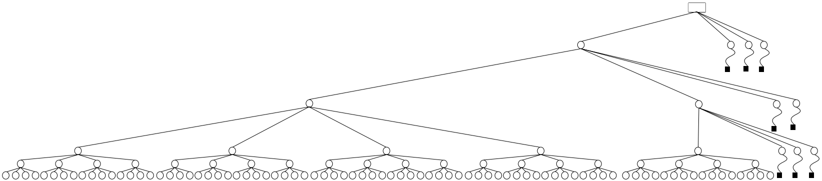
node ← current node;
while *node is not the root* **do**
    update node decision variables;
    node ← parent of node;
end



Decision-tree after the explained iteration.

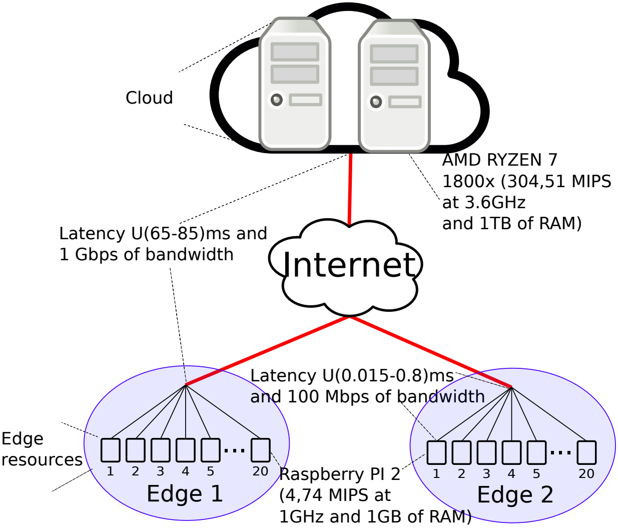


Decision-tree after consuming the iteration budget.



Simulation - Experimental Setup

- Developed atop **OMNET++**.
- Network topology based on (Hu et al., 2016).



Evaluated RL algorithms:

- **MCTS-UCT** (Sutton and Barto, 2018) basic version of the Monte-Carlo Tree Search with UCT.
- **TDTS-Sarsa(λ)** (Vodopivec, Samothrakis and Ster, 2017) creates intermediary rewards for each operator movement and employs them when estimating the reward.

Parameters of the RL algorithms:

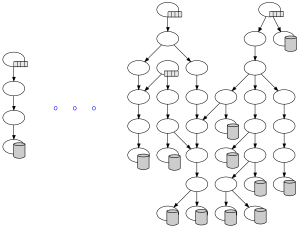
- The operator reconfiguration is triggered when the execution reaches **300** seconds or all application paths have processed **500** messages; whichever comes last.
- Computational budget of **10,000** iterations.

Simulation - Performance Comparison

- State-of-the-art:
  - Traditional approach (**Cloud**).
  - **LB** (Taneja and Davy, 2017), which considers CPU, memory, and bandwidth constraints to obtain the operator placement.

Simulation - Evaluated Applications

Example of application graphs



Operator Parameters

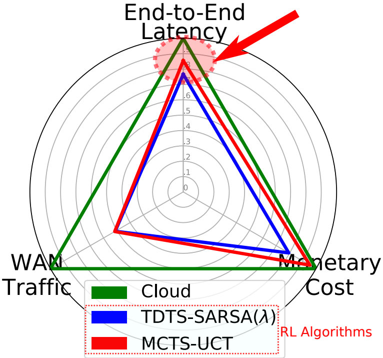
Parameter	Value	Unit
cpu	1000-10000	Instructions per second
Data compression rate	0-90	%
mem	100-7500	bytes
Input message size	100-2500	bytes
Selectivity	0-90	%
Input message rate	1000-10000	Number of messages
window size*	1-100	Number of messages

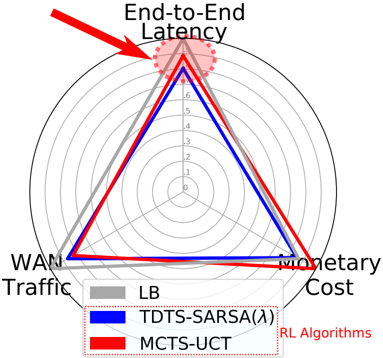
* only for stateful operators.

- **Eleven application graphs** with single and multiple application paths obtained using a Python library (*Generic graphs* 2019).
- **20%** of operators are stateful.
- Data sources and sinks are placed on the edges, except for the sink of the longest application path (cloud).

Alexandre da Silva Veith, Felipe Rodrigo de Souza, Marcos Dias de Assunção, Laurent Lefevre, Julio Cesar Santos dos Anjos. *Multi-Objective Reinforcement Learning for Reconfiguring Data Stream Analytics on Edge Computing*. ICPP 2019.

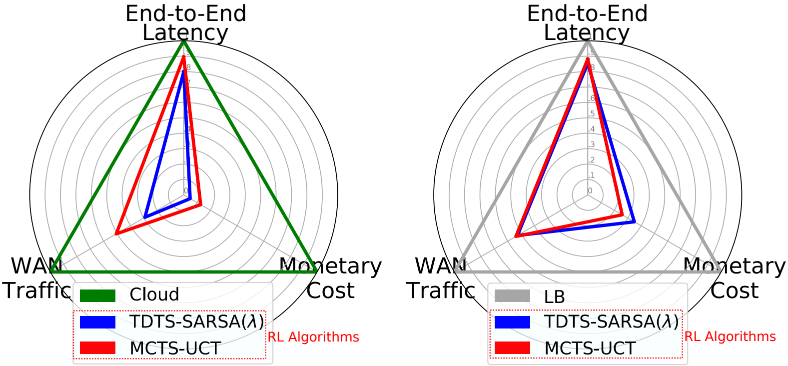
Scenario 1: End-to-End Latency with weight equal to 1.





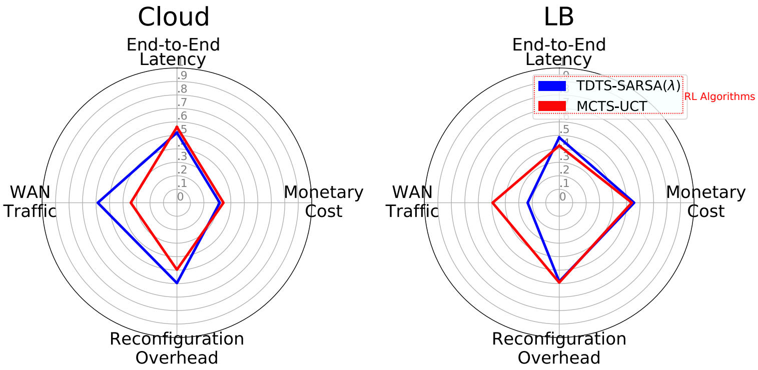
Our implementation achieves over **20% better end-to-end latency**, and reduces the WAN traffic by over **50%** and the monetary cost by **15%** when comparing to Cloud approach.

Scenario 2: Cloud and LB with weights for end-to-end latency, monetary cost and WAN traffic equal to 0.33.



Our implementation **places operators closer to data sources and sinks** reducing the WAN traffic and monetary cost.

Scenario 3: 0.4, 0.2, 0.2, and 0.2 weights for end-to-end latency, monetary cost, WAN traffic and reconfiguration overhead, respectively.



Our implementation reduces the end-to-end latency in over **45%**, the WAN traffic in over **40%**, the monetary cost in over **50%** and the reconfiguration overhead in over **50%**.

Table of Contents

Contributions

- Initial operator placement (application configuration):
  - A placement model with multiple operator properties.
  - Two placement strategies for reducing the end-to-end latency.
  - Simulations that demonstrate an end-to-end latency reduction in over 50%.
  - Real testbed evaluations, which show that our strategy outperforms the state-of-the-art and a random approach by over 38%.

Contributions Cont.

- Application reconfiguration:
  - A model for employing RL algorithms for the DSP application reconfiguration.
  - A multi-objective optimisation approach, which covers metrics for reducing the data traffic on Internet links, the monetary cost, the reconfiguration overhead and the end-to-end latency.
  - Simulations using synthetic and generic applications, which demonstrate that our approach improves in average all performance metrics by 50%.

Open Doors

- Investigation of failure methods.
  - Computing resources can fail during the application execution.
  - Information can be lost (stateful operators) during failures.
  - Fault tolerant methods must be investigated (e.g., check-pointing or operator replication) for reducing the impact of additional resource demand.

Open Doors

- Investigation of failure methods.
  - Computing resources can fail during the application execution.
  - Information can be lost (stateful operators) during failures.
  - Fault tolerant methods must be investigated (e.g., check-pointing or operator replication) for reducing the impact of additional resource demand.
- Lack of techniques for reducing the search space of RL algorithms in the (re)configuration domain.
  - An action space with all computing resources can result in a long time for converging in good reward solutions.
  - Other machine learning methods should be proposed.

Thank you!

Journal:

- Marcos Dias de Assunção, **Alexandre da Silva Veith** and Rajkumar Buyya. *Distributed data stream processing and edge computing: A survey on resource elasticity and future directions*. Journal of Net. and Computer Applications 2018.

International conferences:

- **Alexandre da Silva Veith**, Marcos Dias de Assunção and Laurent Lefevre. *Monte-Carlo Tree Search and Reinforcement Learning for Reconfiguring Data Stream Processing on Edge Computing*. SBAC-PAD 2019.
- **Alexandre da Silva Veith**, Felipe Rodrigo de Souza, Marcos Dias de Assunção, Laurent Lefevre, Julio Cesar Santos dos Anjos. *Multi-Objective Reinforcement Learning for Reconfiguring Data Stream Analytics on Edge Computing*. ICPP 2019.
- Eduard Gibert Renart, **Alexandre da Silva Veith**, Daniel Balouek-Thomert, Marcos Dias de Assunção, Laurent Lefevre and Manish Parashar. *Distributed Operator Placement for IoT Data Analytics Across Edge and Cloud Resources.* CCGrid 2019.
- **Alexandre da Silva Veith**, Marcos Dias de Assunção and Laurent Lefevre. *Latency-Aware Placement of Data Stream Analytics on Edge Computing*. ICSOC 2018.

Book chapter:

- **Alexandre da Silva Veith** and Marcos Dias de Assunção. *Apache Spark*. Encyclopedia of Big Data Technologies 2018.

Hu, Wenlu, Ying Gao, Kiryong Ha, Junjue Wang, Brandon Amos, Zhuo Chen, Padmanabhan Pillai and Mahadev Satyanarayanan. "Quantifying the Impact of Edge Computing on Mobile Applications". In: *7th ACM SIGOPS Asia-Pacific Wksp on Systems*. APSys '16. Hong Kong, Hong Kong: ACM, 2016, 5:1–5:8. ISBN: 978-1-4503-4265-0.

Taneja, M. and A. Davy. "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm". In: *IFIP/IEEE Symp. on Integrated Net. and Service Management (IM)*. May 2017, pp. 1222–1228.

Foroni, Daniele, Cristian Axenie, Stefano Bortoli, Mohamad Al Hajj Hassan, Ralph Acker, Radu Tudoran, Goetz Brasche and Yannis Velegrakis. "Moira: A goal-oriented incremental machine learning approach to dynamic resource cost estimation in distributed stream processing systems". In: *Proceedings of the International Workshop on Real-Time Business Intelligence and Analytics*. ACM. 2018, p. 2.

Ni, L., J. Zhang, C. Jiang, C. Yan and K. Yu. "Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets". In: *IEEE IoT Journal* PP (2017), pp. 1–1. ISSN: 2327-4662.

Mortazavi, Seyed Hossein, Mohammad Salehe, Carolina Simoes Gomes, Caleb Phillips and Eyal de Lara. "Cloudpath: A Multi-tier Cloud Computing Framework". In: *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*. SEC '17. San Jose, California: ACM, 2017, 20:1–20:13. ISBN: 978-1-4503-5087-7. DOI: 10.1145/3132211.3134464. URL: http://doi.acm.org/10.1145/3132211.3134464.

Peng, Boyang, Mohammad Hosseini, Zhihao Hong, Reza Farivar and Roy Campbell. "R-Storm: Resource-Aware Scheduling in Storm". In: *16th Annual Middleware Conf.* Middleware '15. Vancouver, BC, Canada: ACM, 2015, pp. 149–161. ISBN: 978-1-4503-3618-5.

Shukla, Anshu, Shilpa Chaturvedi and Yogesh Simmhan. "RIoTBench: An IoT benchmark for distributed stream processing systems". In: *Concurrency and Computation: Practice and Experience* 29.21 (2017), e4257.

Gelly, Sylvain and David Silver. "Monte-Carlo tree search and rapid action value estimation in computer Go". In: *Artificial Intelligence* 175.11 (2011), pp. 1856–1875.

Russo, Gabriele Russo, Matteo Nardelli, Valeria Cardellini and Francesco Lo Presti. "Multi-Level Elasticity for Wide-Area Data Streaming Systems: A Reinforcement Learning Approach". In: *Algorithms* 11.9 (2018), p. 134.

Azure IoT Hub. 2019. URL: https://azure.microsoft.com/en-us/services/iot-hub/.

AWS IoT Core. 2019. URL: https://aws.amazon.com/iot-core/pricing/.

Sutton, Richard S. and Andrew G. Barto. *Reinforcement Learning: An introduction*. MIT press, 2018.

Vodopivec, Tom, Spyridon Samothrakis and Branko Ster. "On Monte Carlo Tree Search and Reinforcement Learning". In: *Journal of Artificial Intelligence Research* 60 (2017), pp. 881–936.

Generic graphs. 2019. URL: https://gist.github.com/bwbaugh/4602818.