北京大学高能效计算与应用中心
Center for Energy-efficient Computing and Applications

Edge-Stream: a Stream Processing Approach for Distributed Applications on a Hierarchical Edge-computing System

Xiaoyang Wang, Zhe Zhou, Ping Han, Tong Meng, Guangyu Sun, Jidong Zhai

Xiaoyang Wang
Peking University, Beijing, China
2020.11.11

Overview

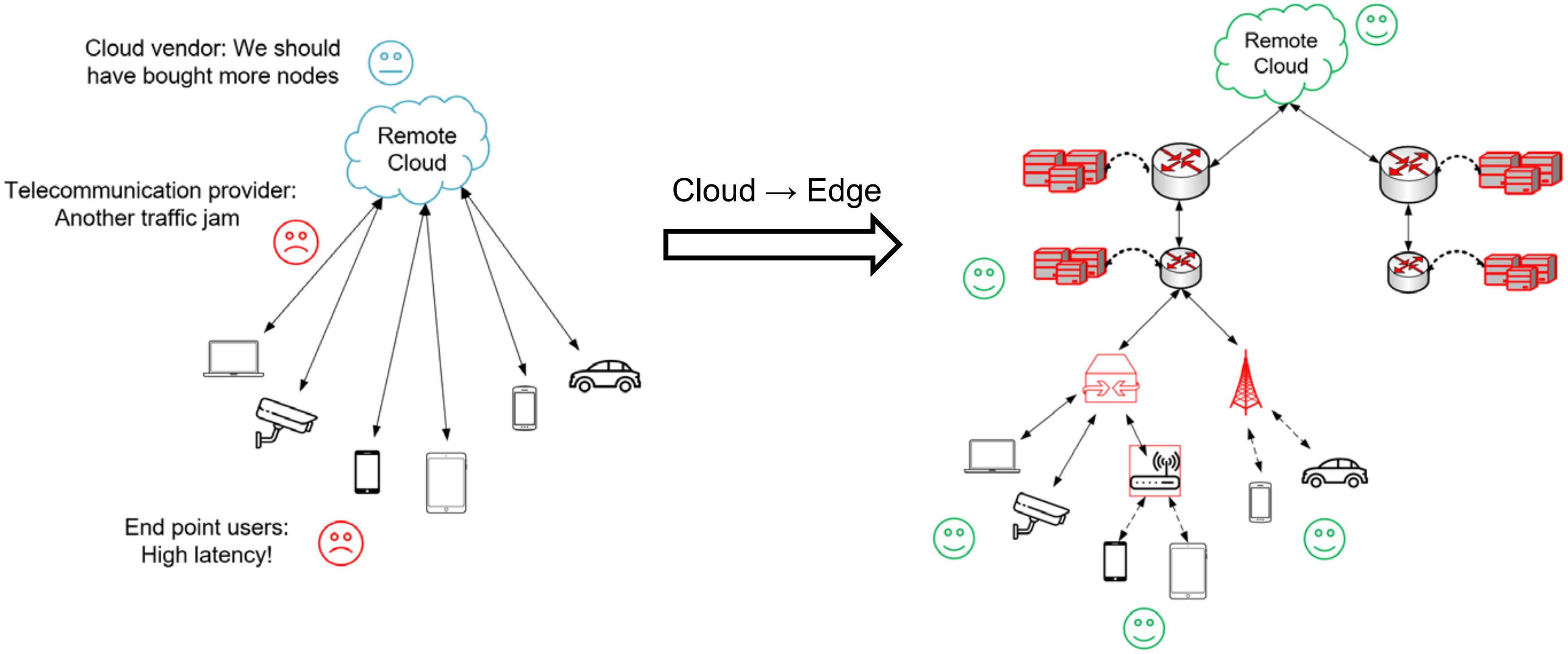
- Motivation
- Edge-Stream Model
- EStream Platform
- Evaluation
- Conclusion

Overview

- Motivation
  - Complexity in programming
  - Various scenarios
  - Collaboration among users
- Edge-Stream Model
- EStream Platform
- Evaluation
- Conclusion

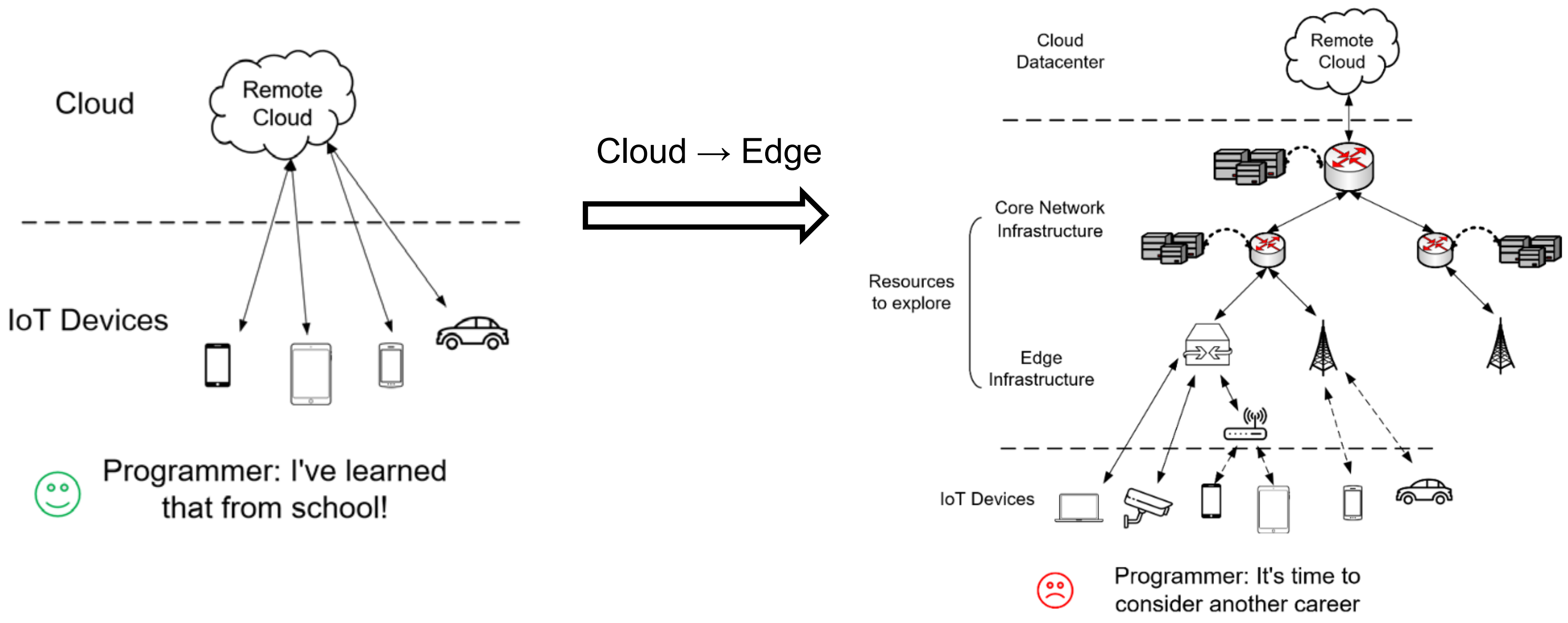
Edge Computing

- **Edge computing** relieves the pressure of cloud and reduces the latency by taking the burden of computation away from remote data center (the **Cloud**) to computation nodes (the **Edge**) near those IoT devices.



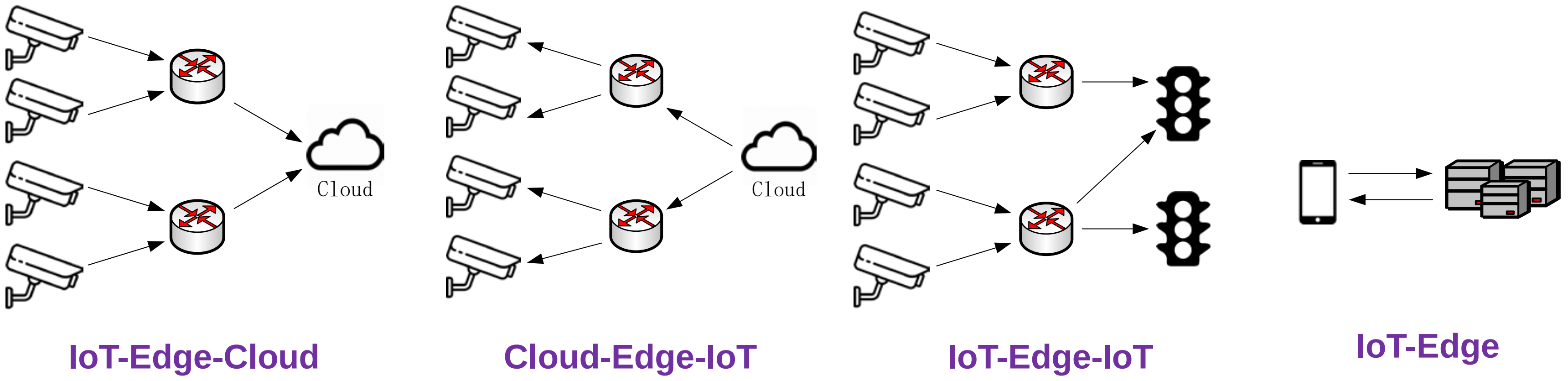
Complexity in programming

- Edge computing introduces **complexity** in developing efficient applications on IoT devices
  - More computation levels are taken into consideration
  - Computation nodes are geo-distributed



Various scenarios

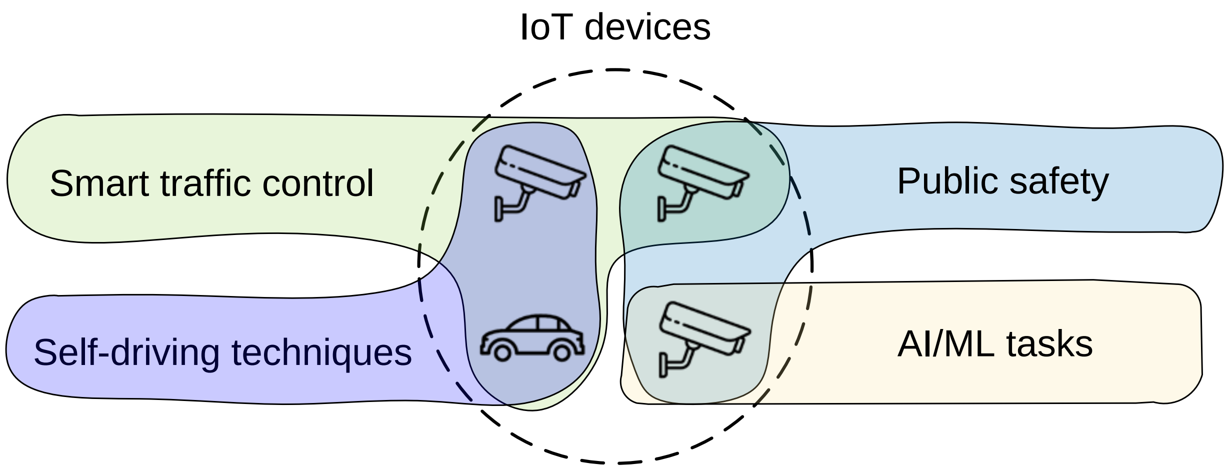
- Different IoT applications vary a lot.



- Most existing research approaches can only handle one of those typical scenarios, while real-world applications always involve multiple of them.

Collaboration among IoT owners

- IoT devices are deployed for multiple purposes.



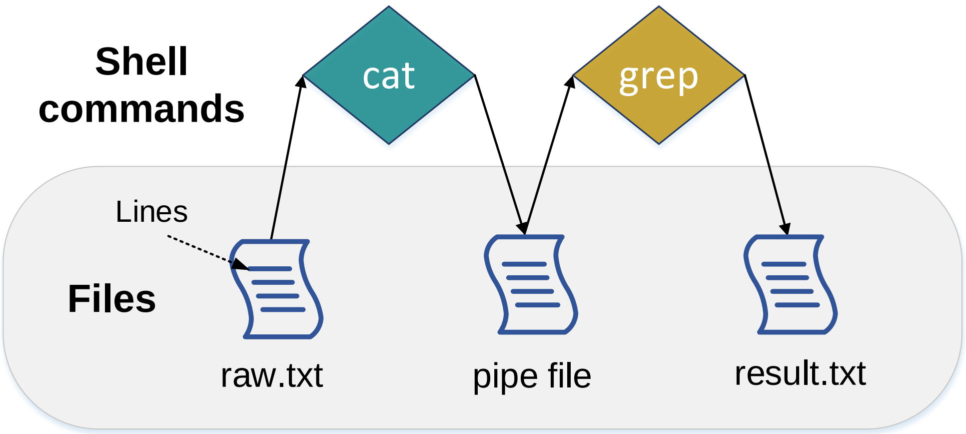
- Data from the same IoT device may be used in different tasks.
- One task may involve different kinds of IoT devices.

Overview

- Motivation
- Edge-Stream Model
  - Software development on Linux
  - Stream and operator design in Edge-Stream model
- EStream Platform
- Evaluation
- Conclusion

Software development on Linux

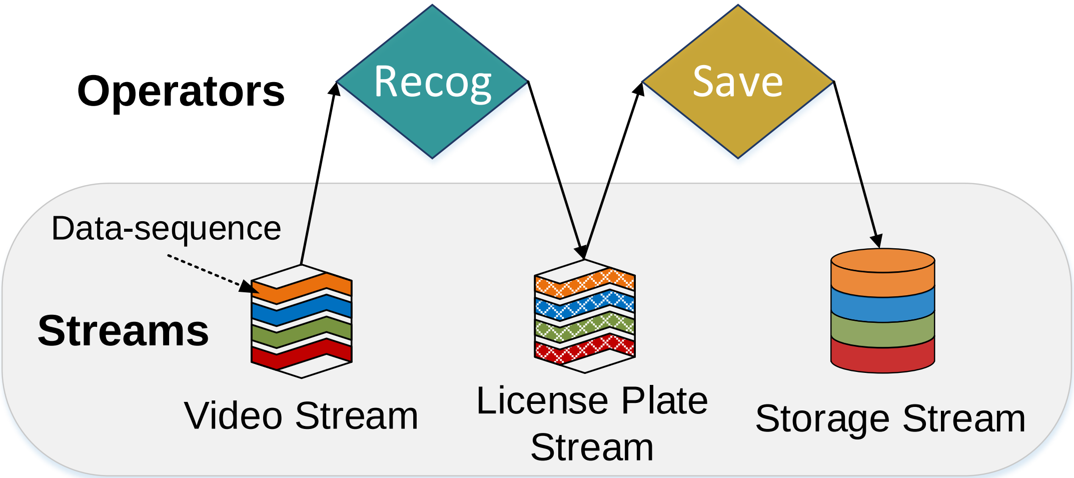
- Files
  - Users manage and share data blocks by file.
- Commands
  - Programmers care more about the “format” instead of the specific “content” of input files when developing.
- Shell scripts
  - Scripts are composed of pre-defined commands.




```
cat raw.txt | grep "hello" > result.txt
```

Edge-Stream: stream-based model for edge computing

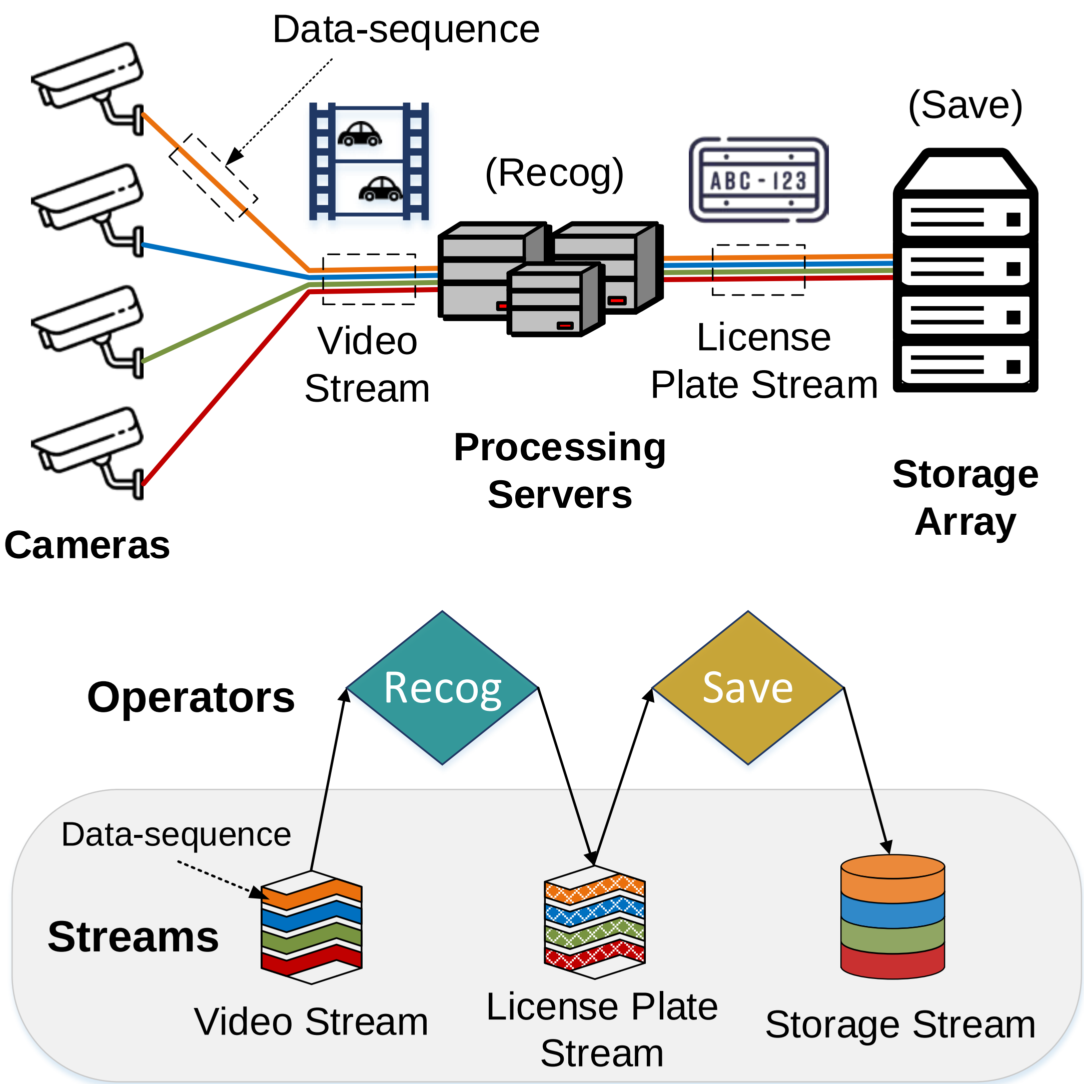
- Streams
  - Users manage and share data sequences by stream.
- Operators
  - Programmers care more about the "format" instead of the specific "content" of input streams when developing.
- Applications
  - Applications are composed of pre-defined operators.

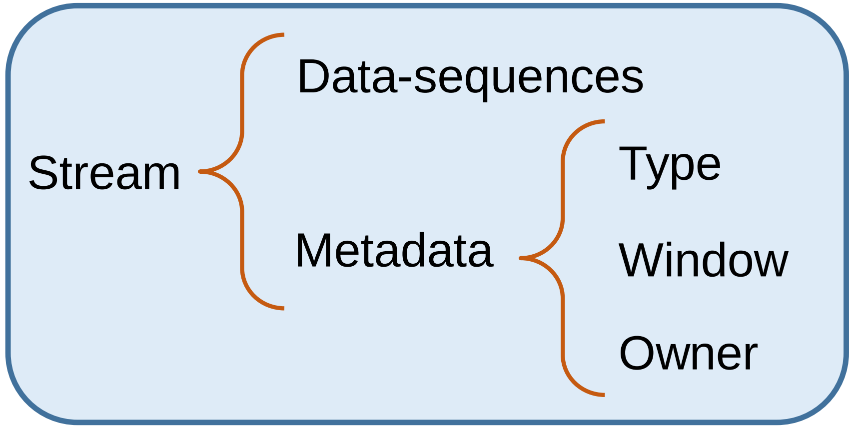


Edge-Stream: stream-based model for edge computing

Physical system

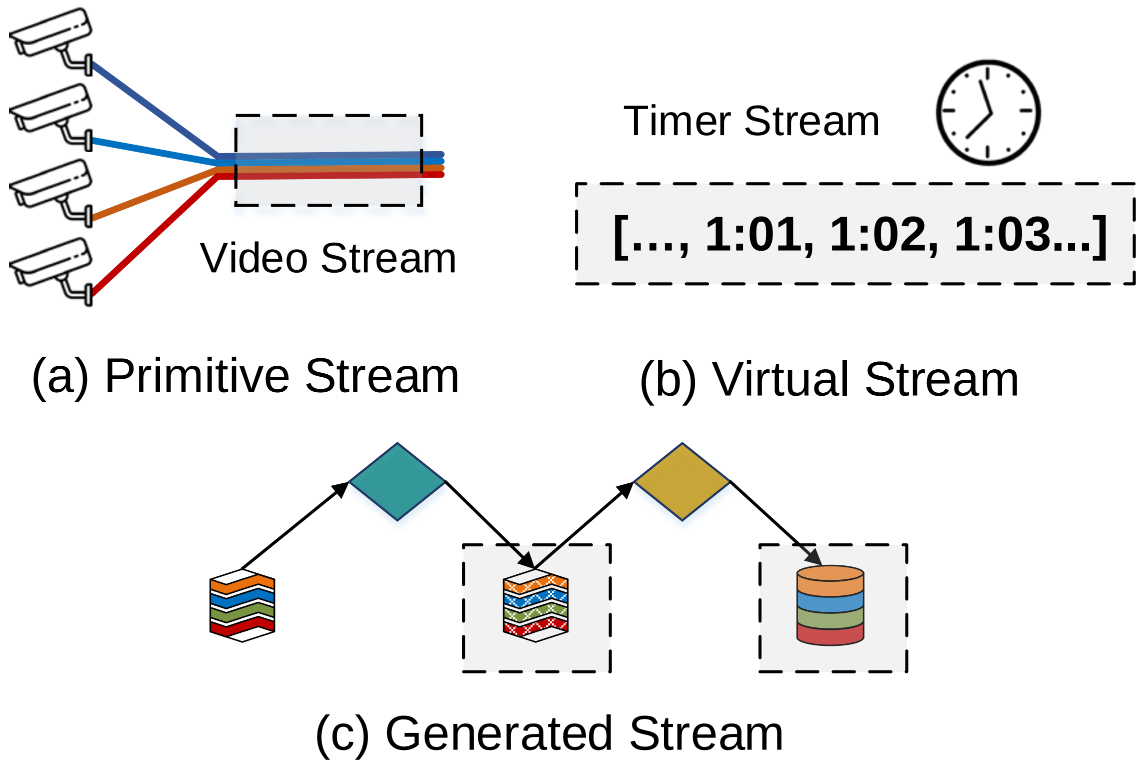
Abstract view





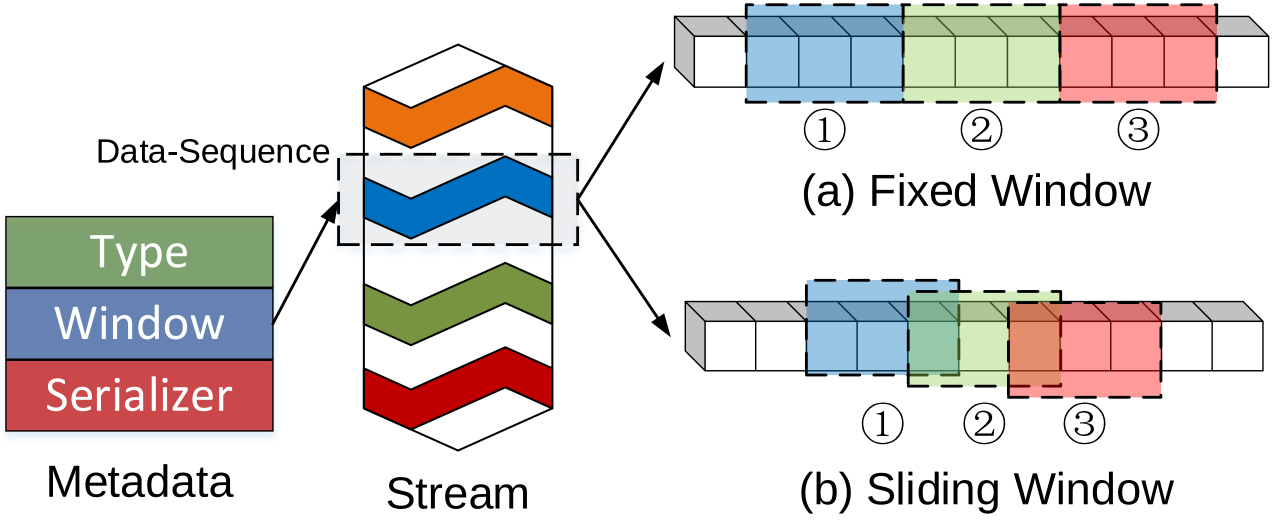
Stream design: types

- Different types of streams
  - Primitive stream: generated directly from endpoint devices.
  - Virtual stream: generated on demand by any node in the system.
  - Generated stream: generated by operators (the input streams are called parent streams).



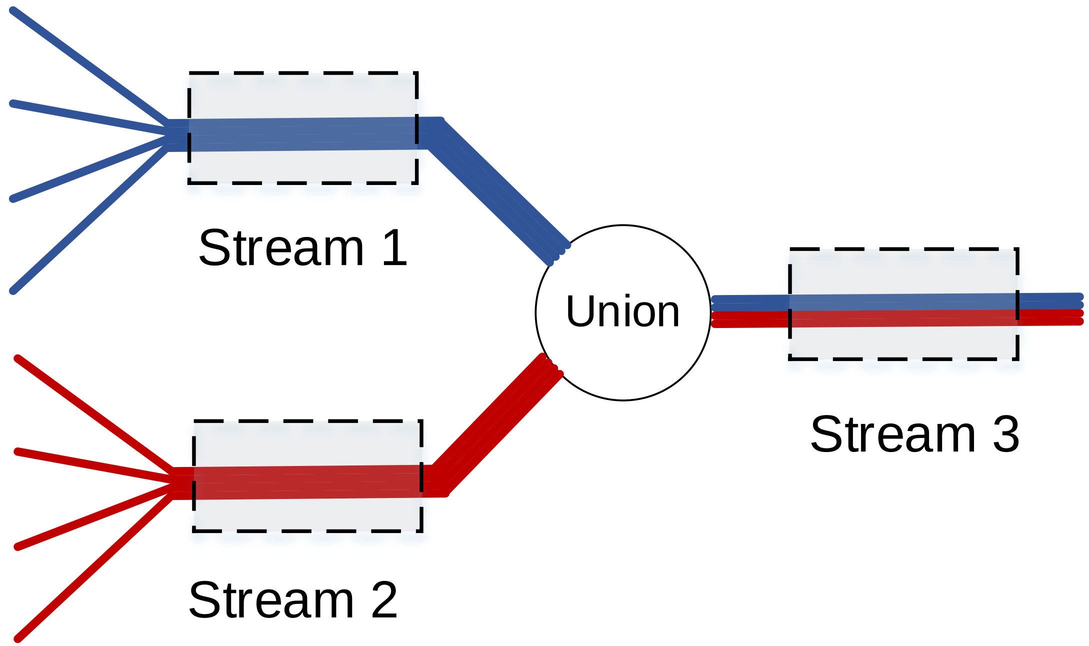
Stream design: windows

- Windows
  - Widely adopted in traditional distributed computing frameworks.
  - Define how data will be aggregated in physical nodes.



Operator design

- Reshaping operators
  - Define how to organize existing data-sequences, **without changing the data inside**.
  - Examples: Union, windowing operations



Operator design

- Computation operators
  - Generate new data from input streams with **functions**.

$$Operator_f(\{a,b,c\}) = \{f(a), f(b), f(c)\}$$

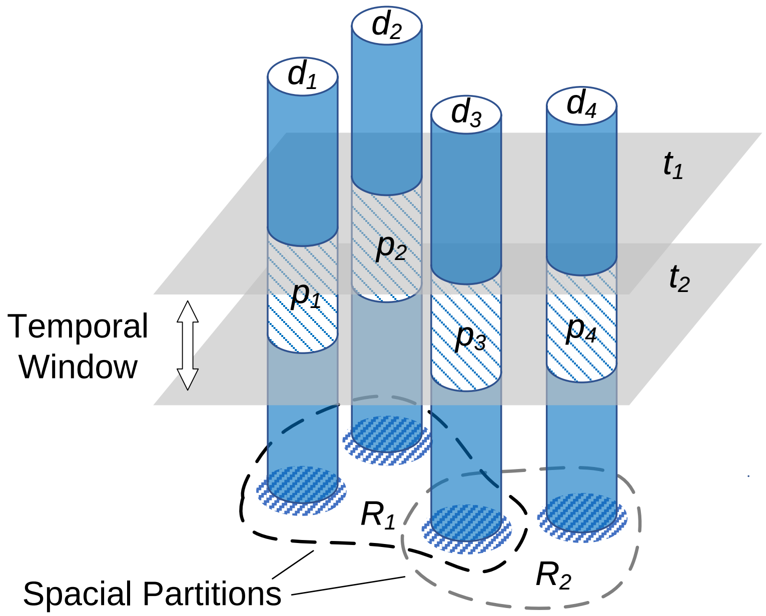
- Functions access data through a standard set of APIs
  - Map-style functions: getNext()
  - Reduce-style functions: getWindow()

```
#include <string>
#include "MyRecogLib"
%%
%in S_video<Picture, null, File>
%out S_plate<std::string, null, JSON>
%%
%{
  auto inPicture = S_video.getNext();
  auto outPlate = PlateRecog(inPicture);
  S_plate.pushItem(outPlate);
%}
```

```
#include <string>
%%
%in S_plate<std::string, fixed, JSON>
%out S_result<int, null, JSON>
%%
%{
  int counter = 0;
  auto plates = S_plate.getWindow();
  for (plate : plates) {
    counter ++;
  }
  S_result.pushItem(counter);
%}
```

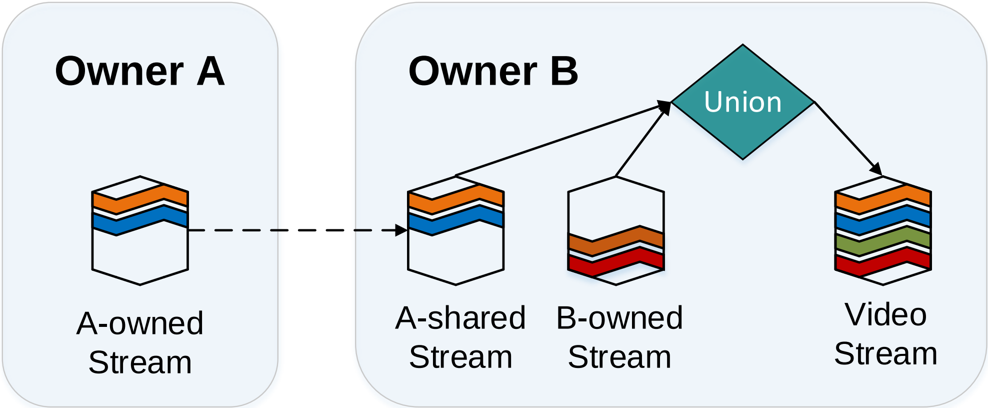
Grouping method

- Reorganize data-sequences
  - Similar to keyBy/GroupByKey transformations in traditional big data frameworks
  - Grouping provides **spacial** partitions (Windows generate **temporal** slices).



Stream sharing

- Each stream has a unique owner.
  - The owner is able to share the stream to other users.
  - Those users are allowed to build new streams from it, but cannot modify or delete the original stream.

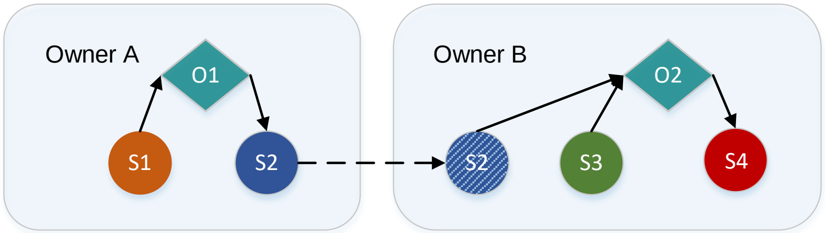


Overview

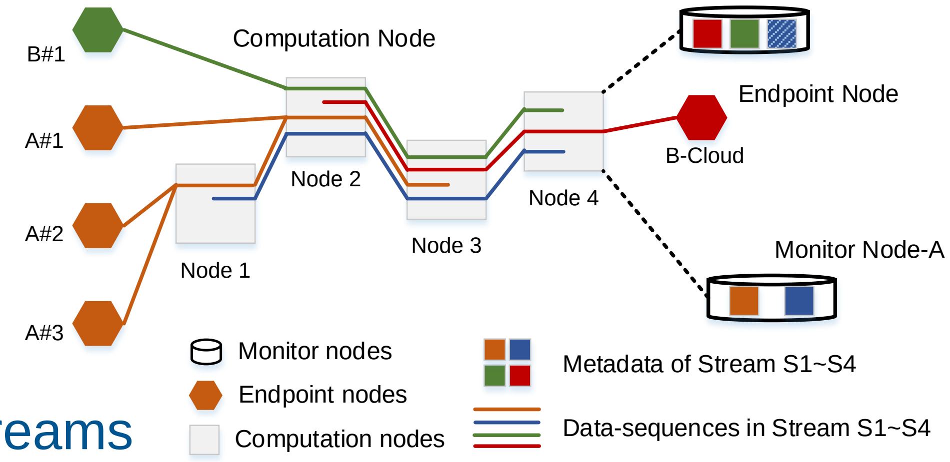
- Motivation
- Edge-Stream Model
- EStream Platform
  - Architecture overview
  - Stream creation
  - Request propagation
  - Decentralized scheduling
- Evaluation
- Conclusion

Architecture overview

- Endpoint node
  - IoT & Cloud
  - Provide primitive streams
- Computation node
  - Provide virtual streams and generated streams
- Monitor node
  - Locates in the cloud, maintaining the metadata of streams
  - Provide services to interact with streams



(a) Job description



(b) Three kinds of nodes in EStream

Stream creation

- Necessary information to create a stream in the system
  - Primitive stream: a list of endpoint devices / a list of areas
  - Virtual stream: its generation algorithm
  - Generated stream: its parent streams

- Find parent streams for a generated stream
  - Ask their monitor nodes for help
    - Primitive stream: locate devices / areas on the list
    - Virtual stream: create it on demand
    - Generated stream: recursively find its parents
  - Caching techniques help to accelerate the procedure

Where does the input data of the stream **come from**

Request propagation

- **Direction** to deliver data-sequences
  - Sinked streams: adopt the intuitive direction towards the sink node
  - Other streams: transmit the result to its monitor node

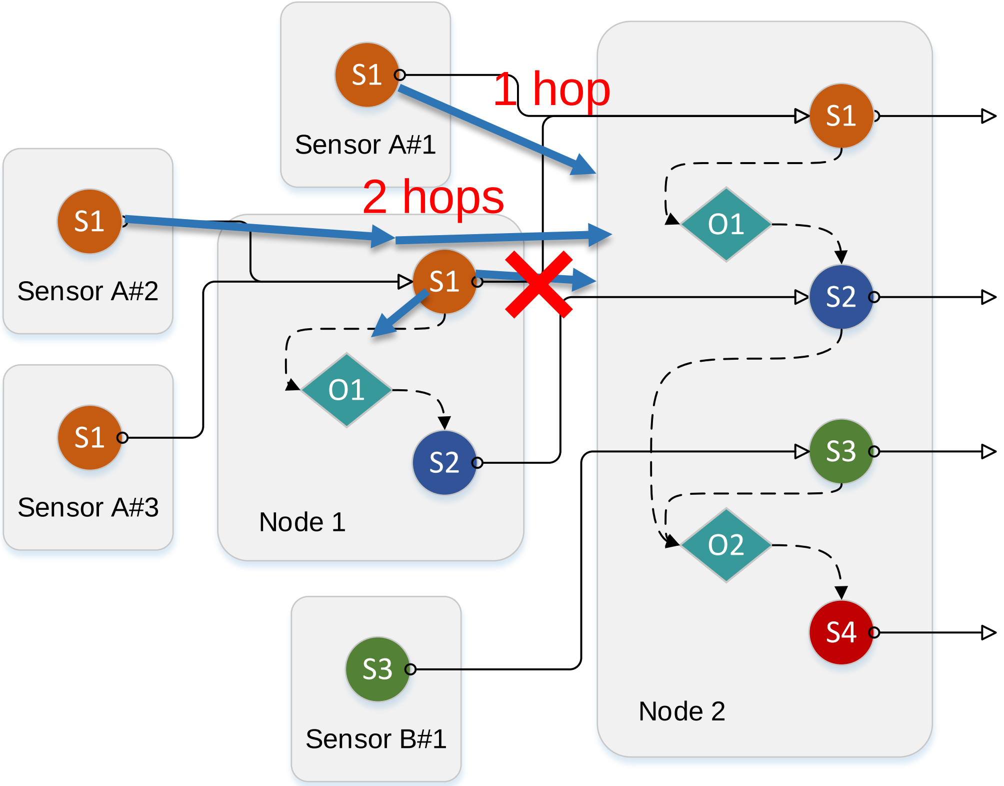
The **direction** for the stream to go

- **Location** that each data-sequence first appears in the system
  - Map-style functions: where the input data first appears in the stream
  - Reduce-style functions: the nearest common ancestor node is used to collect the data in the same window

The **location** where the stream is generated

Decentralized scheduling

- Target: balance the lifetime of packages in the same stream.
  - Nodes prefer to compute data packages with a larger transmission latency in the same stream.
  - The algorithm selects to push the computation pressure backwards to the data sources.
  - Merge computation to improve the locality of data.



- Both of data sources and sinks have “attraction” to the workload

Find more nodes to do the computation for the stream

Overview

- Motivation
- Edge-Stream Model
- EStream Platform
- Evaluation
- Conclusion

Experimental setup

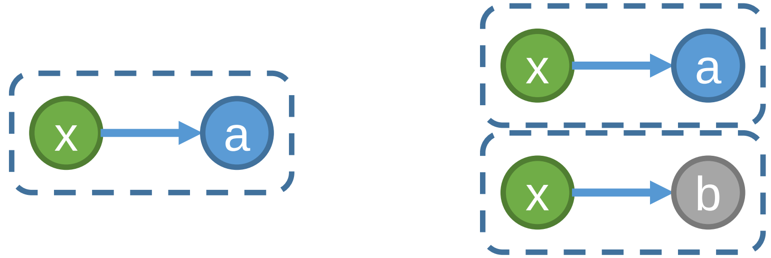
- Default network topology

	Access latency (ms)	Number of nodes	Profiling machine
Cloud	110	1	Workstation with Xeon 6148 CPU, 256GB RAM and 4 GTX-2080Ti GPUs.
Router	15	10	PC with i7-6700K CPU, 32GB RAM and a GTX-1080ti GPU.
Access Point	5	100	PC with i7-6700K CPU, 32GB RAM
IoT device	0	1000	Raspberry Pi 4B with 4GB RAM

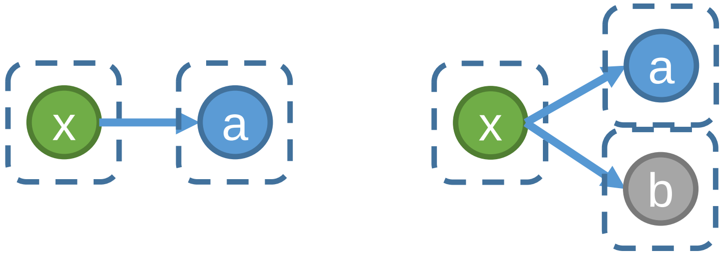
- Test case: Smart traffic system
  - Job x: vehicle detection
  - Job a: license plate numbers recognition (long-lasting job)
  - Job b: vehicle attributes recognition (emergent task)

Benefits of stream sharing

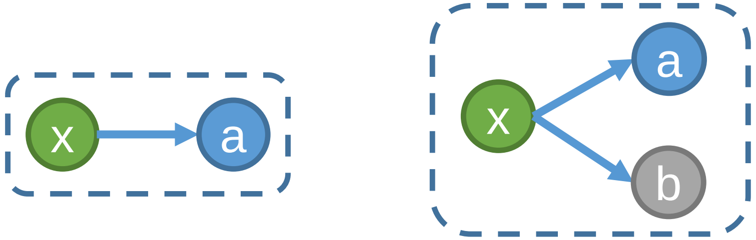
- Change the job from {x+a} to {x+ab}



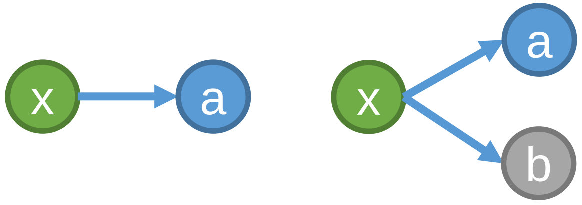
Flink 0: compute twice



Flink 1: plan in advance



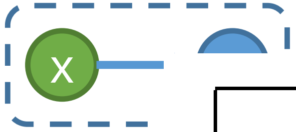
Flink 2: stop and restart

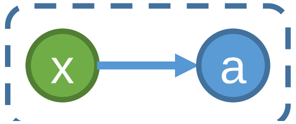


EStream: stream-level sharing

Benefits of stream sharing

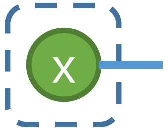
- Change the job from {x+a} to {x+ab}

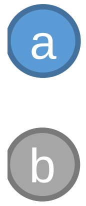




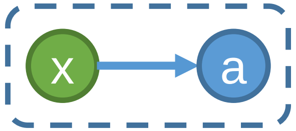
Flink 0: compute twice

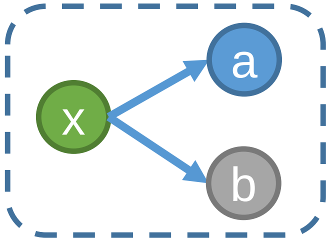
	Flink 0	Flink 1	Flink 2	**EStream**
Latency x+a (t_0)	295 ms	295 ms	295 ms	**295 ms**
Latency x+a (t_1)	341 ms	336 ms	449 ms	**305 ms**
Latency x+b (t_1)	312 ms	307 ms	276 ms	**276 ms**
Energy (t_0)	47 J	47 J	47 J	**47 J**
Energy (t_1)	85 J	67 J	77 J	**64 J**





aring



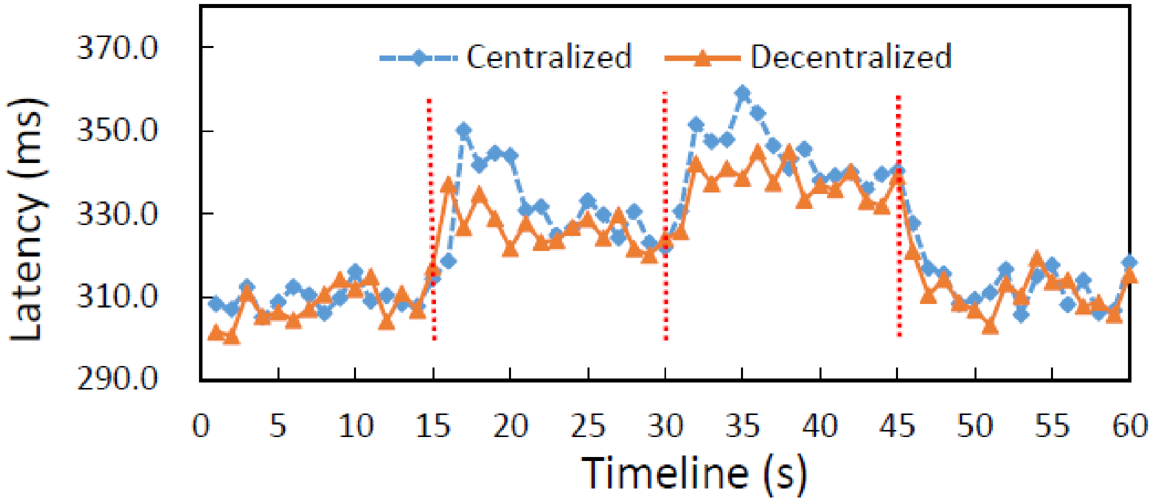


Flink 2: stop and restart

Decentralized scheduling

- Evaluation settings
  - 4 cloud data-centers & 50 routers
  - On average: **IoT** ↔ 1 **access point** ↔ 2.9 **routers** ↔ **cloud**

- Four stages:
  - Initial job: x+a
  - Change to: x+a&b
  - Duplicate the job
  - Restore the initial settings



Conclusion

- Edge-Stream: Stream-centric computation model
  - Support various IoT scenarios
  - Hide the complicated network topology from developers
  - Simplify the collaboration among IoT owners

- EStream: a prototype realization of Edge-Stream
  - Help to verify the benefit from the new model
  - Provide a practical scheduling method

Thank you!

- Name: Xiaoyang Wang
- Email: yaoer@pku.edu.cn
- Address: Peking University, Beijing, China