Active-Code Reloading in the OODIDA Platform

12 June 2018

Gregor Ulm, Emil Gustavsson, Mats Jirstrand
Fraunhofer-Chalmers Research Centre
for Industrial Mathematics, Gothenburg, Sweden

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

OODIDA

Paper:

OODIDA: On-board/Off-board Distributed Data Analysis for Connected Vehicles

Gregor Ulm
Fraunhofer-Chalmers Research
Centre for Industrial Mathematics
Gothenburg, Sweden
gregor.ulm@fcc.chalmers.se

Emil Gustavsson
Fraunhofer-Chalmers Research
Centre for Industrial Mathematics
Gothenburg, Sweden
emil.gustavsson@fcc.chalmers.se

Mats Jirstrand
Fraunhofer-Chalmers Research
Centre for Industrial Mathematics
Gothenburg, Sweden
mats.jirstrand@fcc.chalmers.se

ABSTRACT

A modern connected vehicle produces dozens of gigabytes of data per hour. Performing centralized data processing within a reasonable amount of time with data generated by just one connected vehicle would be taxing enough. Doing the same with a fleet of hundreds or thousands of vehicles, on the other hand, is impossible. A related problem is the complexity involved in not only distributing one task to edge devices, but having them perform multiple distributed large-scale tasks concurrently. Consequently, there is a need for both decentralized data processing and task management. The OODIDA (On-board/off-board distributed data analysis) platform tackles both issues. It has been designed for handling concurrent distributed data analysis. Its key feature is the ability to concurrently execute multiple distributed data analysis tasks on overlapping subsets of client devices. Tasks can be finite or infinite. An example of the former is training of a machine learning model with a fixed number of iterations, while an example of the latter is stream processing on each client.

The OODIDA platform is modular and language-agnostic, with the exception of the central server process, which is based on the actor model and implemented in Erlang. Assignments are specified in JSON. Results can be specified in an arbitrary format, for instance as plain text files. This means that the central server process is able to accept assignments from any application that is able to output JSON; even manual specification is possible. In addition, applications on the client can be implemented in an arbitrary programming language. As a consequence, the modular approach of the OODIDA platform leads to great flexibility.

KEYWORDS

Distributed computing, Concurrent computing, Applied functional programming, Data analysis, Distributed Data Processing, Erlang

1 INTRODUCTION

While distributed systems are a commonplace phenomenon in the computing industry, they constitute a nascent technology in many traditional industries. One such example is the transportation industry. As personal and commercial vehicles are increasingly network-connected and equipped with on-board units, which are general-purpose computers in all but name, we explored approaches to distributed data analysis for two international corporations in the automotive industry. Both face the issue of handling a *bona fide* deluge of data. As transferring dozens of gigabytes per vehicle per hour via the network and performing centralized analysis is not feasible, we therefore developed a system for distributed data analysis that is based on the actor model (cf. Section 4.2). This was further modified to enable the concurrent execution of distributed data analysis tasks.

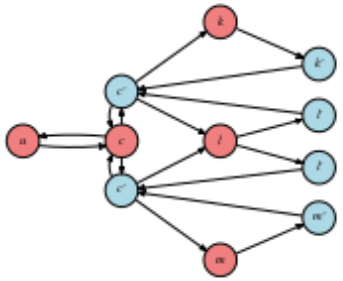


Figure 1: Overview of the OODIDA framework

OODIDA is a modular platform for concurrent distributed data analysis. It has been created for the automotive domain with its

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Overview

- OODIDA: Context
- OODIDA: System Details
- OODIDA: Sample Use Cases
- Limitations (Problem)
- Active-Code Reloading (Solution)

The OODIDA Platform in Context

Context

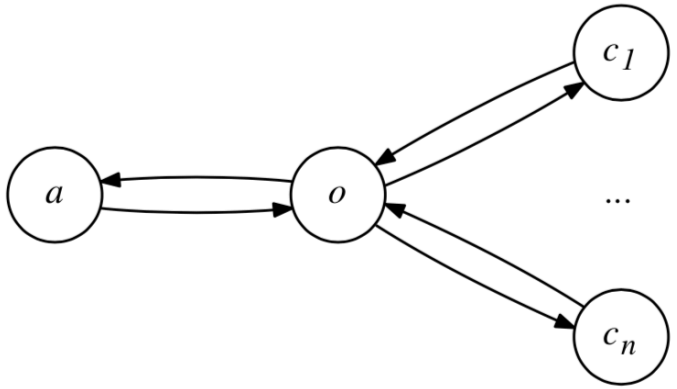
- Big Data in the automotive industry
- Currently ~50 GB/hour generated per car
  - Can be easily increased (more sensors, higher sampling rate)
- Large commercial fleets
- Current main paradigm, data is processed as a batch after-the-fact
- Real-time capabilities lacking
- Goal: Platform for (pseudo) real-time analytics
  - This is the OODIDA platform

Problem

- Quintessential big data problem
- Volume: dozens of gigabytes/hour per car
  - Transfer to central server infeasible
- Velocity: we want timely insights
  - Storage-and-process paradigm unsuitable
- Variety: myriad of signals and sensors to observe
  - One-size-fits-all approach won't work
- Privacy: very detailed profiling possible with big data
  - Not possible if most data never leaves the client
  - GDPR may apply

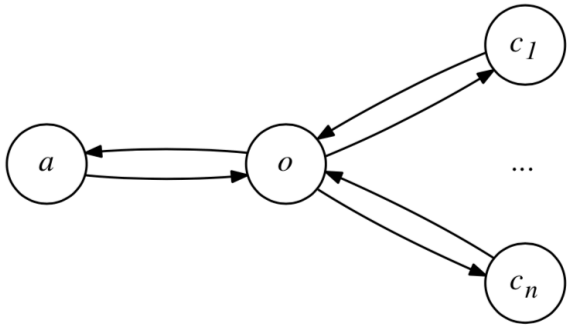
OODIDA Overview

- Data analysis platform written in Erlang and Python
- Interaction with hardware -> cyber-physical system
  - On-board unit on clients (c_i)
- o: OODIDA platform
- a: analyst (one for illustration)
- OODIDA is both a simulator and a real-world system



FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Problem: Usability



- Different skills in big data analytics
- Analyst/Data Scientist: working with data, applying algorithms, maybe implementing algorithms
  - Python (libraries!)
- Software Engineer: creating and maintaining the platform
  - Erlang, some Python
- Thus, different levels of access to OODIDA

Role of the Analyst

- Defining an assignment for clients
  - Data collection
  - Result can be final data or the input for further local processing
- Example assignment:

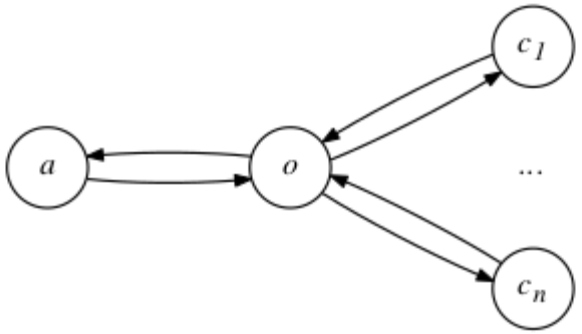
```
# task 1: all vehicles
spec = { "id"          : 1
       , "name"        : "Measurement Task"
       , "description" : "Average of velocity"
       , "mode"        : "Measurement"
       , "clients"     : "all"
       , "priority"    : None
       , "signal"      : "velocity"
       , "onboard"     : "average"
       , "offboard"    : "collect"
       , "frequency"   : 5000
       , "num_reports" : 1
       }
```

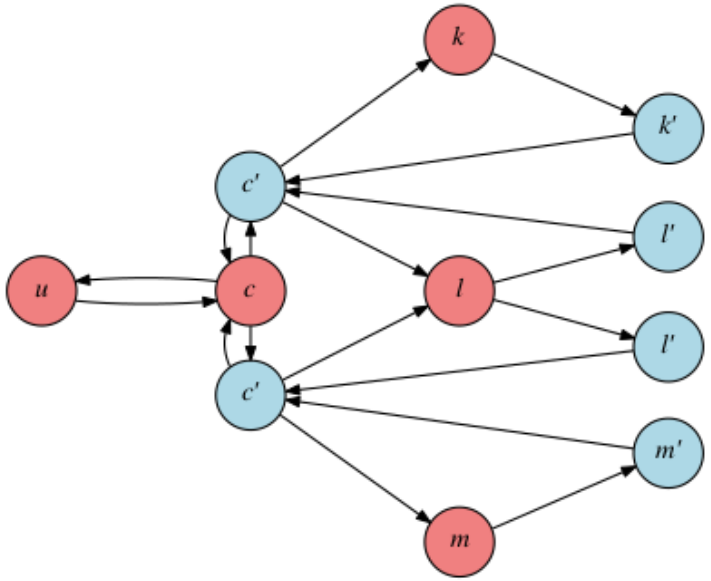
(In comparison, the Software Engineer ensures that the Analyst can do their work.)

System Details

OODIDA in Context

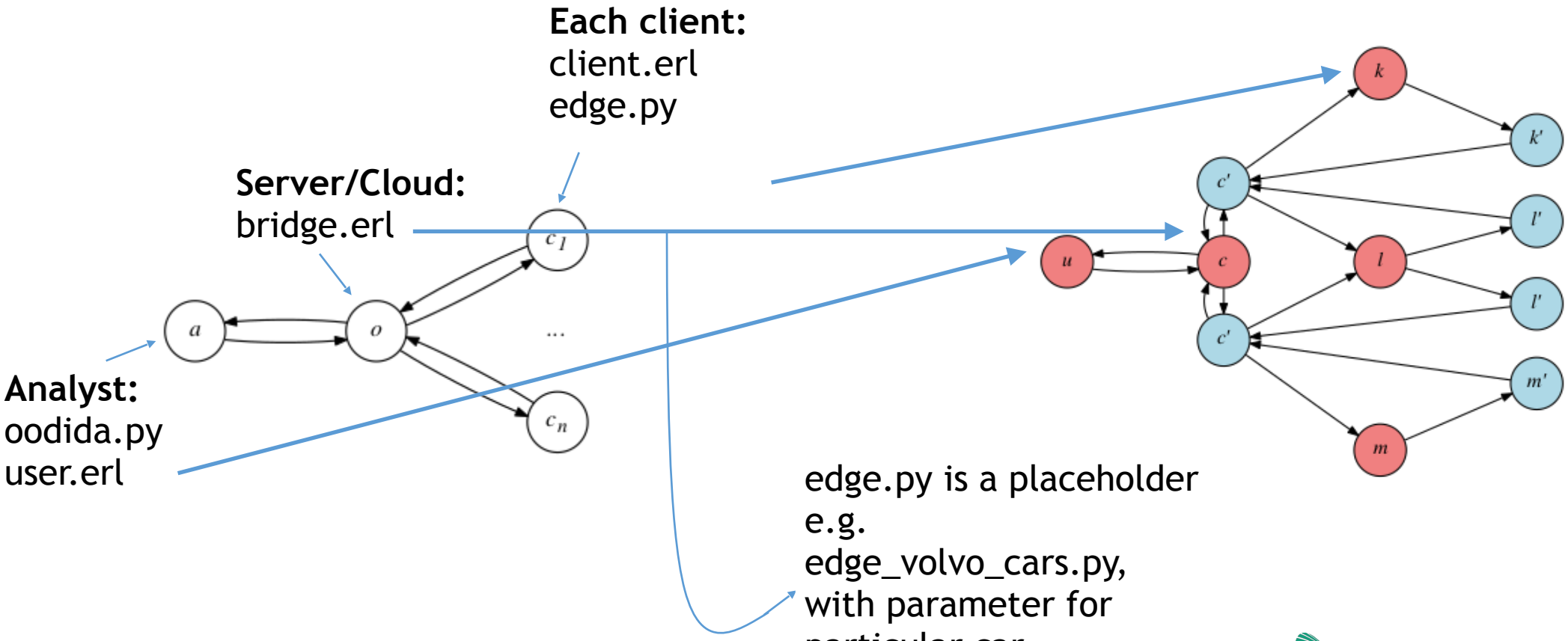
- Analyst
- OODIDA
- Clients





FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Modularity of the System

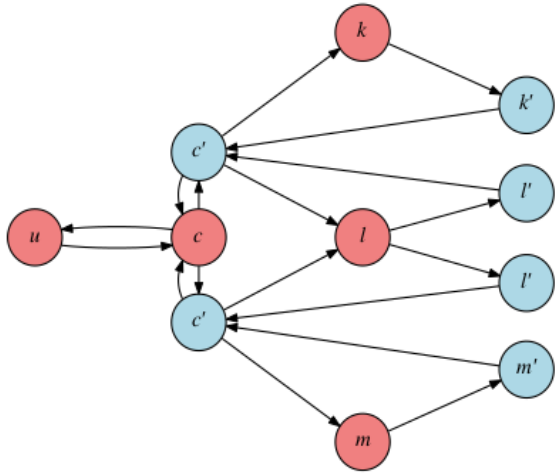


FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

OODIDA in Detail

- Analyst (u)
- Cloud (c)
- Clients (k, l, m

- Red nodes: permanent
- Blue nodes: temporary (so-called assignment handlers/task handlers)



Workflow (single-round assignment):
. u waits for assignment file
. if file received: u sends data to c
. c spawns assignment handler c' (top)
. c' (top) connects to clients k, l
. Clients k, l spawn their own (task) handler
. handler on clients write assignment as JSON, await completion
. external process takes over, does assigned task
. when completed, task handler on client reads results file, forwards to c'
. after all results have been received, c' sends aggregate to c
. c forwards results to u, writes to file

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

A Sample Assignment in Detail

Goal: make the job of the user

```
# task 1: all vehicles
spec = { "id"          : 1
       , "name"        : "Measurement Task"
       , "description" : "Average of velocity"
       , "mode"        : "Measurement"
       , "clients"     : "all"
       , "priority"    : None
       , "signal"      : "velocity"
       , "onboard"     : "average"
       , "offboard"    : "collect"
       , "frequency"   : 5000
       , "num_reports" : 1
       }
```

```
import lib_user.oodida as o
o.createAssignment(spec)
```

(That's it!)

Notes:

- The OODIDA library verifies that the provided specification is correct (structure, data types, range of values)
- priority not yet implemented

Grammar of an Assignment

```
# task 1: all vehicles
spec = { "id"          : 1
       , "name"        : "Measurement Task"
       , "description" : "Average of velocity"
       , "mode"        : "Measurement"
       , "clients"     : "all"
       , "priority"    : None
       , "signal"      : "velocity"
       , "onboard"     : "average"
       , "offboard"    : "collect"
       , "frequency"   : 5000
       , "num_reports" : 1
       }
```

```
"id"          : positive integer
"name"        : arbitrary string, can be empty
"description" : arbitrary string, can be empty
"mode"        : one of: "Measurement"       ,
                        "Filter"            ,
                        "Federated Learning",
                        "EventDetection"    ,
                        "ApplyCalculation"
"clients"     : list of positive integers or string "all"
"priority"    : positive integer
"signal"      : one of: "velocity"    ,
                        "acceleration",
                        "fuel"
"onboard"     : one of: 'average',
                        'median' ,
                        'collect'
. "offboard"    : dto.
. "frequency"   : positive integer or string "indefinite"
. "num_reports" : positive integer
```

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

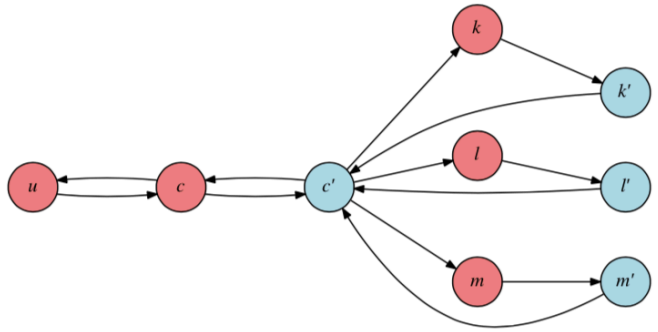
Flexibility of Assignments

- Select all vehicles, or a subset thereof
- Each client executes 0 to *n* tasks concurrently (no clear upper bound)
- Tasks can have finite duration or be indefinitely long
- Tasks have an arbitrary starting time
- Tasks can consist of 1 to *m* iterations
- Results of iteration *i* can be used as input for iteration *i* + 1, e.g. result of *i* of *f(x, d)* is *x'*, iteration *i* + 1 is performed as *f(x', d')* – new data and updated model *x'*

Sample Use Cases

Monitoring

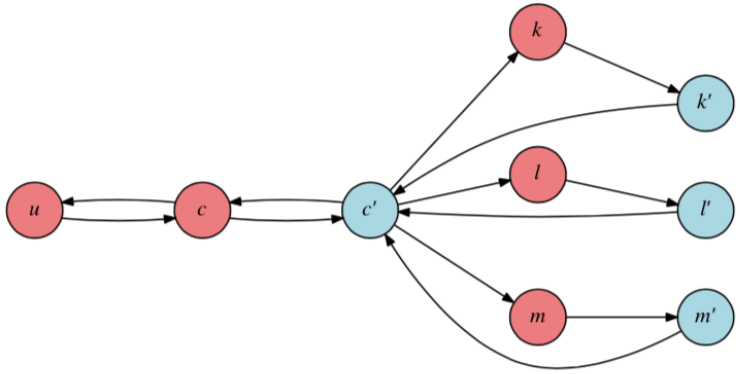
- "Monitor status of sensor X, inform user if threshold exceeded"
- Specify sensor and threshold in assignment
- Client: collects values, sends values that exceed threshold to cloud (runs indefinitely long)



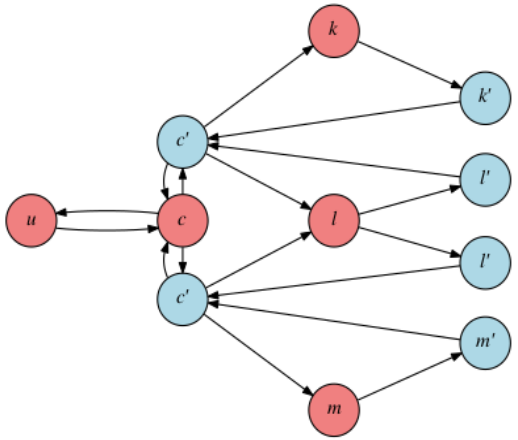
FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Sampling

- "Create representative sample of data produced by sensor X"
- Specify sensor and sample rate in assignment



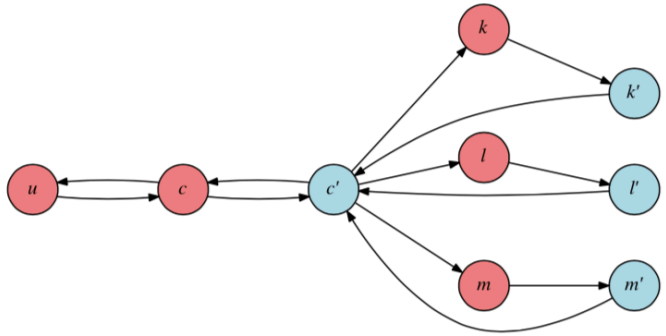
Can also run concurrently with other task (each assignment executed on two clients):



FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Batch Processing

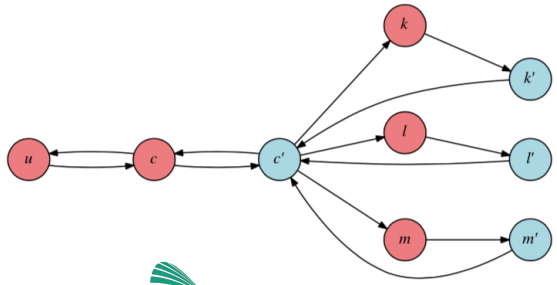
- "Process data generated by sensor X, using algorithm A"
- Specify amount of data points etc. in assignment
- Results are sent to cloud and processed further, maybe just collected



FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Stream Processing

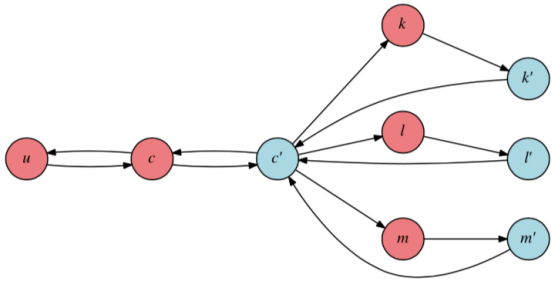
- "Process data generated by sensor X, using algorithm A"
- Specify amount of data points etc. in assignment
  - **Specify number of iterations and send update to cloud after each iteration**
- Stream is modeled as a sequence of batches
- **The shorter the interval, the closer you get to real-time stream processing (of course this is not *real* stream proces**



FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

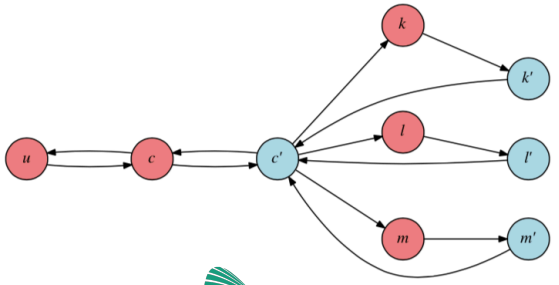
MapReduce

- (I assume you all know MapReduce)
- Let's look at the basic word count example:
  - client: map *(word, 1)* and reduce *(word, count)*
  - server: aggregates all *(word, count)* pairs to *(word, total count)*



Distributed Machine Learning

- "Federated Learning" (misnomer because members of a federation are independent; clients in FL are not)
- initialize global model, send to clients
- clients train their copy of the global model with local data and send local model to server
- server produces new global model
- continues until stopping criterion is met



FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

Limitations (Problem)

Limitations of the Platform

- No easy way to update client code
- Have to redeploy on client devices
  - Shut down client, deploy, restart
  - This terminates ongoing analytics tasks!
- Also: deployment is semi-permanent
  - Removing code likewise requires redeployment
  - Thus, experimentation discouraged

Workaround

- Use the Erlang core of OODIDA to send client code as data
- Client (Erlang) reads data, saves it
- Afterwards, client process (Python) treats it as executable code

Active-Code Reloading (Solution)

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

How it works (for the user)

- Define a Python function
- In principle arbitrary, but right now, almost all our operations on the client are performed on lists of floating-point numbers
- Function call to update “custom function”, e.g.
```
import lib_user.code_update as c
f = "custom_code.py“
c.code_update(f)
```
- Right now, user has to ensure that his code is syntactically correct; will be automated

How it works (for the user)

- Afterwards, user can specify custom code in assignments

```
# task 1: all vehicles
spec = { "id"          : 1
       , "name"        : "Measurement Task"
       , "description" : "Average of velocity"
       , "mode"        : "Measurement"
       , "clients"     : "all"
       , "priority"    : None
       , "signal"      : "velocity"
       , "onboard"     : "average"
       , "offboard"    : "collect"
       , "frequency"   : 5000
       , "num_reports" : 1
       }
```

Replace with “custom”!

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS

How it works (under the hood)

- Library lib_user.code_update treats Python code as data (string)
- Creates JSON file, which is picked up by OODIDA user process
- User process sends update to cloud, cloud disseminates custom code to all clients
- Custom code written to file on each client
- With a new assignment/task, external client process (py) responds to specification of “onboard” computation
- If “custom”, client process reads custom code and executes it with provided input
  - Limitation: Code reloading in Python doesn’t play nicely with global state; thankfully, that doesn’t affect us

What you can do

- Experiment:
  - Execute experimental algorithms on client, without committing
- A/B Testing in parallel:
  - ½ of clients receive custom code A, other ½ custom code B
  - (Instead of sequential testing)
- All, while keeping ongoing tasks alive

What you (deliberately) can't do

- Trivial to add support for multiple custom code functions
- Simple approach: small number of slots, e.g. custom_1 to custom_n
- Problem: don't want users to rely too much on custom code
  - Should be uses temporarily, not as a workaround for the proper deployment process

Acknowledgments

- Vinnova
- Volvo Cars Corporation
- Volvo Group Trucks Technology
- Chalmers University of Technology
- Alkit Communications

FRAUNHOFER CHALMERS
RESEARCH CENTRE FOR INDUSTRIAL MATHEMATICS