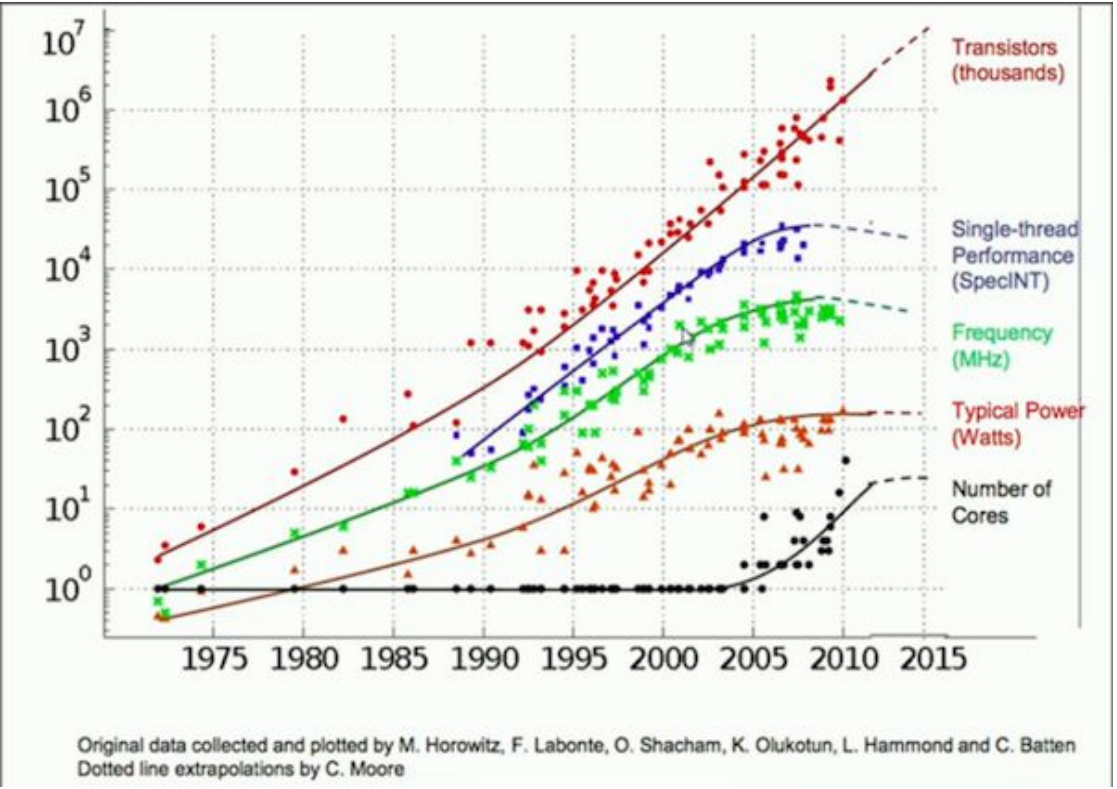
Compute First Networking: Distributed Computing meets ICN

Michał Król[1], Spyridon Mastorakis[2], Dave Oran[3], Dirk Kutscher[4]

[1]University College London/UCLouvain
[2]University of Nebraska, Omaha
[3]Network Systems Research & Design
[4]University of Applied Sciences Emden/Leer

Introduction

Why Distributed Computing?



- Moore's law is failing
- First Pentium 4 processor with 3.0GHz clock speed was introduced back in 2004
- Macbook Pro 2016 has clock speed of 2.9GHz
- Adding more core to the processor has its cost too
- The most reliable way to speed things up is to use multiple CPUs/machines

source:https://medium.com/@kevalpatel2106/why-should-you-learn-go-f607681fad65

Compute First Networking

- Joint optimization of computing and networking
- Taking into account location of the data
- Applications decomposed into small, mobile components
- Constant adaptation to changing environment

Related work

- Multiple Distributed Computing frameworks
  - But usually ignore location of the data
- Function Chaining solutions
  - But usually lack the ability to adapt to changing environment
- Mobile Edge Computing frameworks
  - Often simply extending the cloud computing concept to specific hosts at the edge

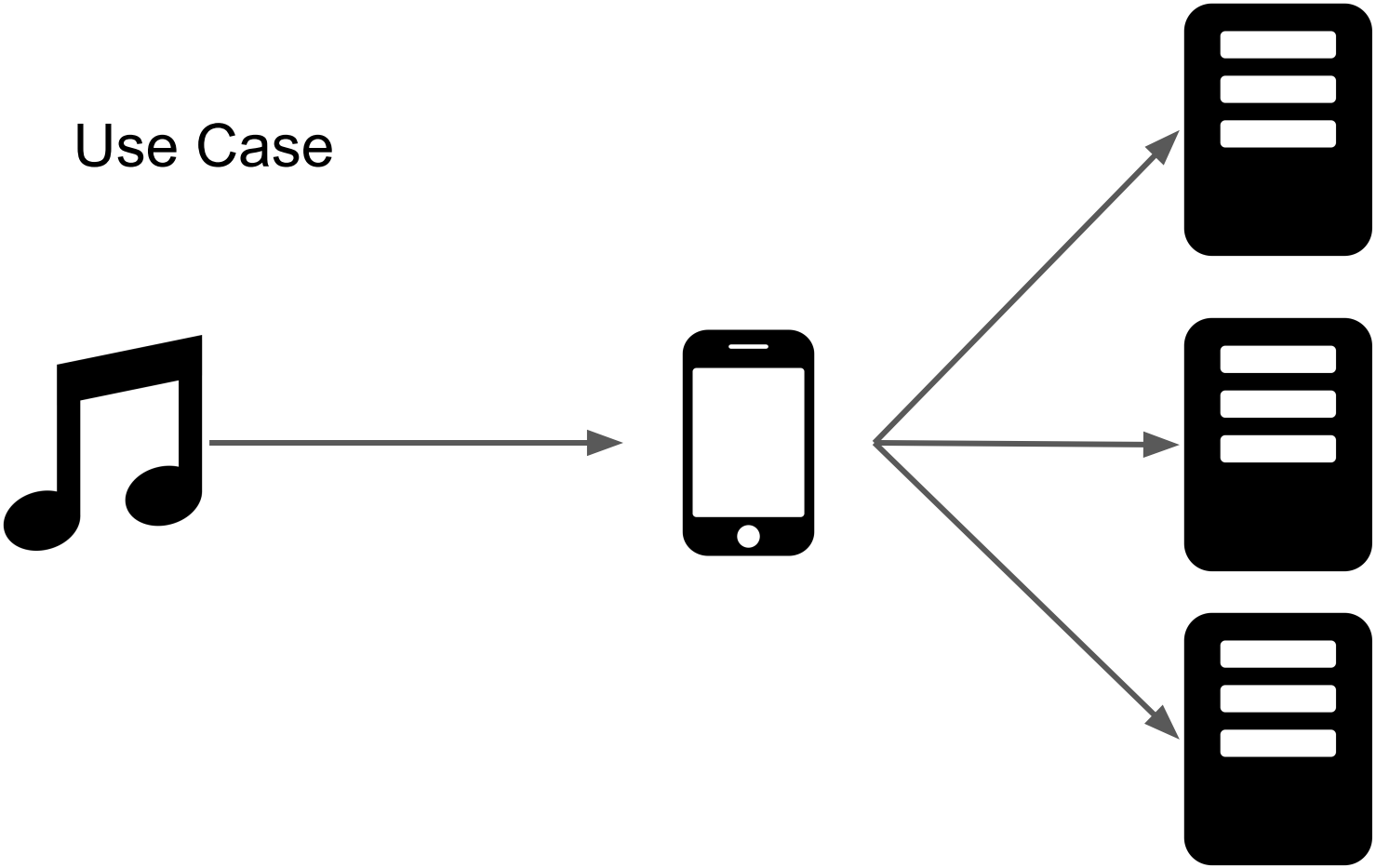
Use Case

- Airport health screening system
- Detect people with highly-infectious pulmonary diseases
- Collect and analyze cough audio samples
- Deployed using commodity mobile phones

Use Case

- Collect samples
- Remove speech
- Detect cough
- Extract cough features (“wetness”, “dryness”)
- Analyse multiple samples

Use Case



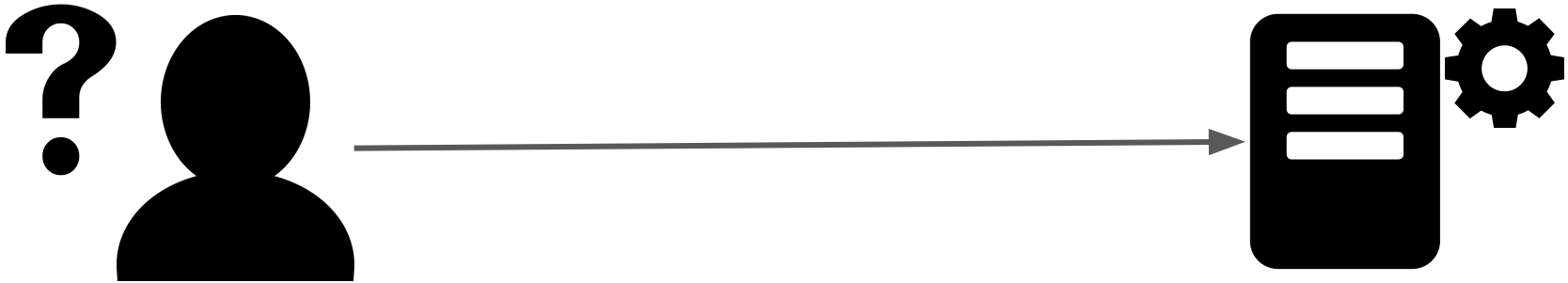
Background

Information Centric Network (ICN)

- Designed for efficient content delivery
- Request (Interest)/ Reply (Data) semantics
- Pushes application level identifiers into the network layer
- Efficient, asynchronous multicast
- Can work on top of layer 2, 3, 4 OSI/ISO protocols

RICE: Remote Method Invocation in ICN

- decouples application and network time
- enables long-running computations through the concept of thunks
- providing additional mechanisms for client authentication, authorization and input parameter passing.
- secure 4-way handshake

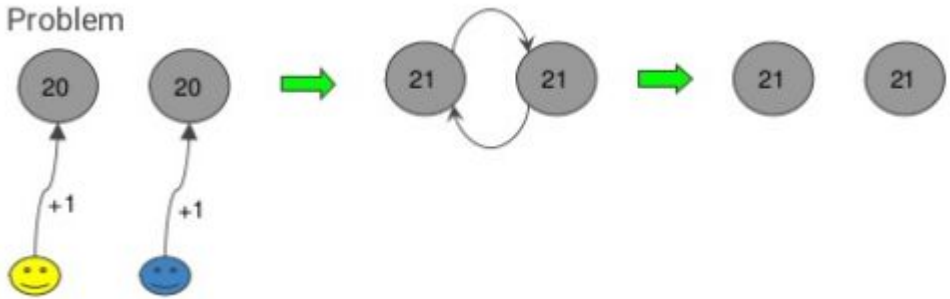


Conflict-Free Replicated Data Types (CRDTs)

- Independent, coordination-free state updates
- Strong eventual consistency guarantees - replicas have a recipe to solve conflicts automatically.
- Enables to satisfy all the CAP theorem properties

Conflict-Free Replicated Data Types (CRDTs)

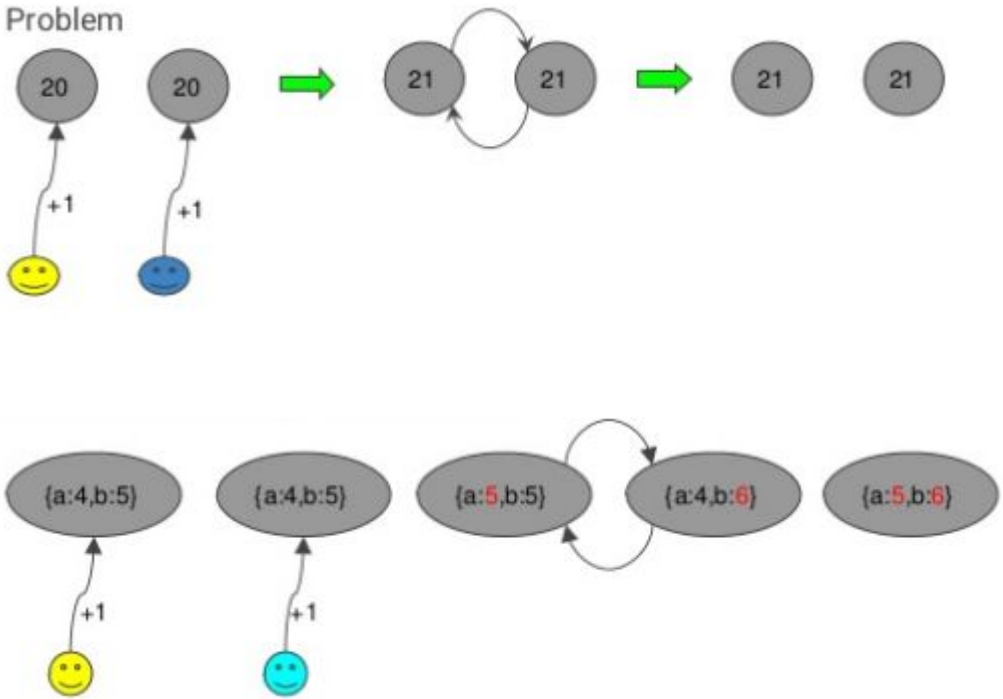
- Independent, coordination-free state updates
- Strong eventual consistency guarantees - replicas have a recipe to solve conflicts automatically.
- Enables to satisfy all the CAP theorem properties



source:https://www.slideshare.net/KirillSablin1/crdt-and-their-uses-se-2016

Conflict-Free Replicated Data Types (CRDTs)

- Independent, coordination-free state updates
- Strong eventual consistency guarantees - replicas have a recipe to solve conflicts automatically.
- Enables to satisfy all the CAP theorem properties



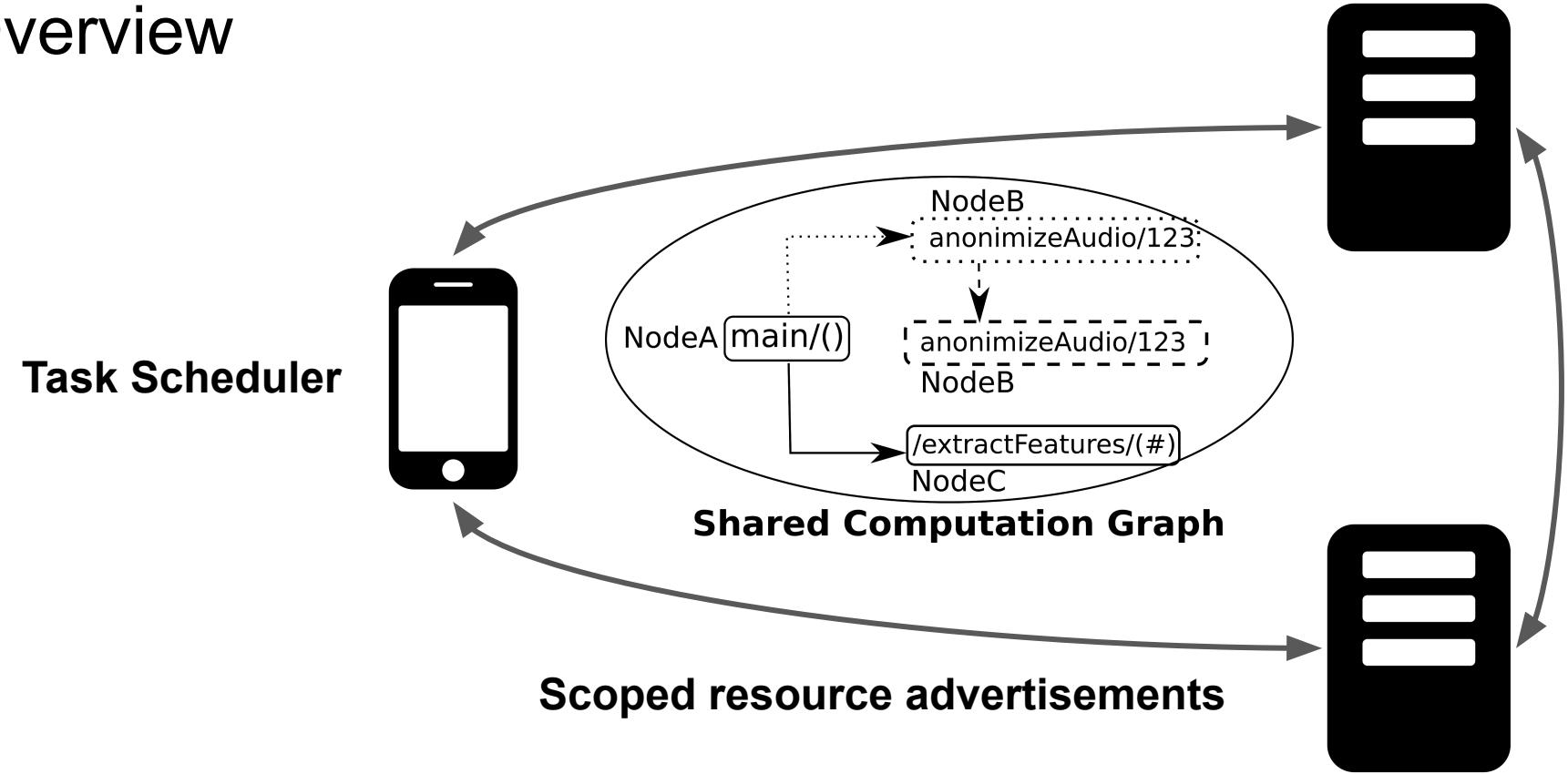
source:https://www.slideshare.net/KirillSablin1/crdt-and-their-uses-se-2016

CFN

Design Goals

- Distributed computing environment for a general purpose programming platform
- Support for both stateless functions and stateful actors
- Flexible load management
- Take into account data location, platform load and network performance
- No major code changes in regard to non-distributed version

Overview


Task Scheduler
NodeB
anonimizeAudio/123
NodeA
main/()
anonimizeAudio/123
NodeB
/extractFeatures/(#)
NodeC
Shared Computation Graph
Scoped resource advertisements

Terminology

- **Program** - a set of computations requested by a user.
- **Program Instance** - one currently executing instance of a program
- **Function** - a specific computation that can be invoked as part of a program.
- **Data** - represents function outputs and inputs or actor internal state.
- **Future** - objects representing the results of a computation that may not yet be computed.
- **Worker** - the execution locus of a function or actor of a program instance

Naming

NodeName: /net1/node1/
Framework prefix: /LA-CFN/
Program Instance: /LA-CFN/EHealth/45/
Computation Graph: /LA-CFN/EHealth/45/graph/
Resource Advertisement: /LA-CFN/resource/

Transparent function: /EHealth/extractFeatures/(#)/
Opaque function: /EHealth/anonimizeAudio/123/

Class method: /EHealth/CoughAnalyzer/f1/(#)/
Class state: /EHealth/CoughAnalyzer/state/#/

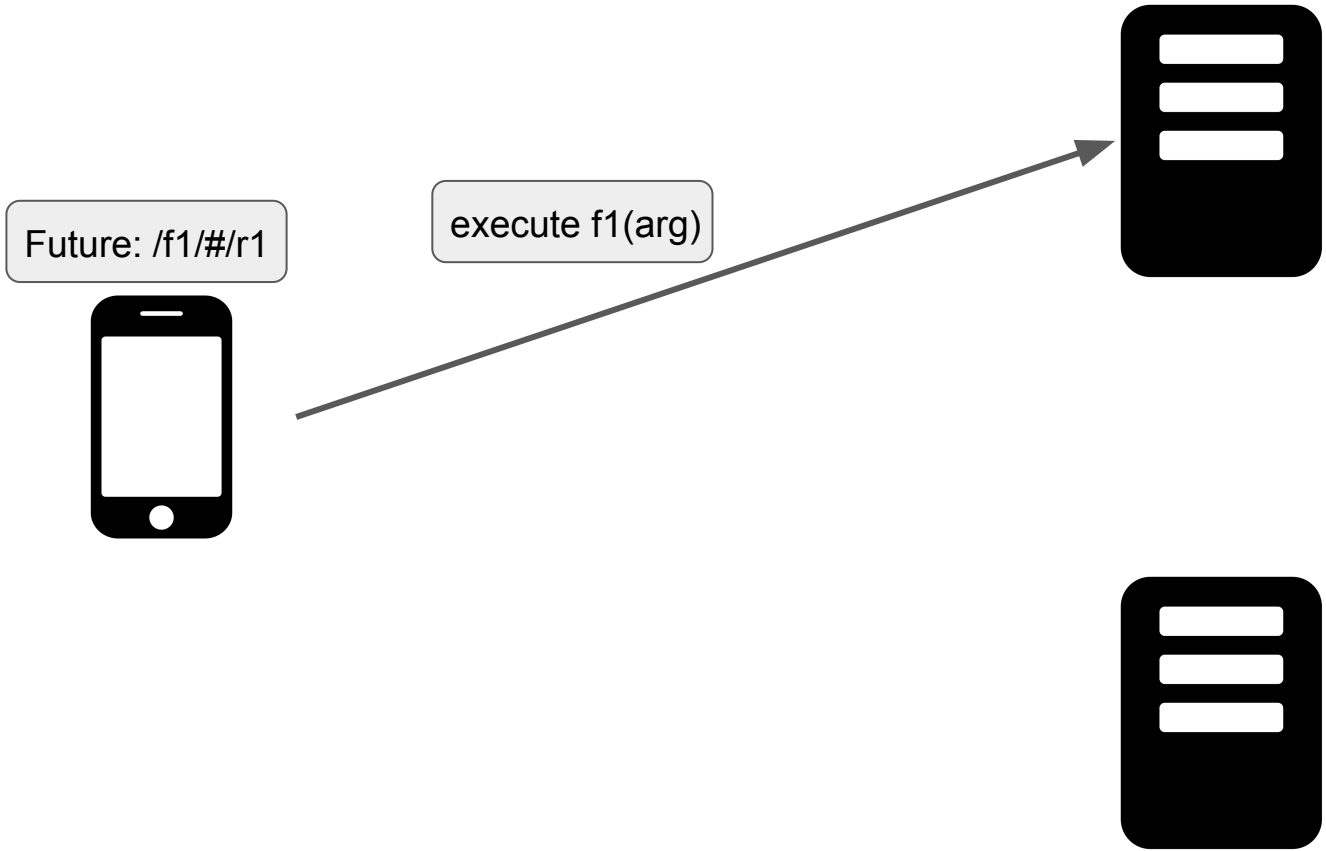
Naming

NodeName: /net1/node1/
Framework prefix: /LA-CFN/
Program Instance: /LA-CFN/EHealth/45/
Computation Graph: /LA-CFN/EHealth/45/graph/
Resource Advertisement: /LA-CFN/resource/

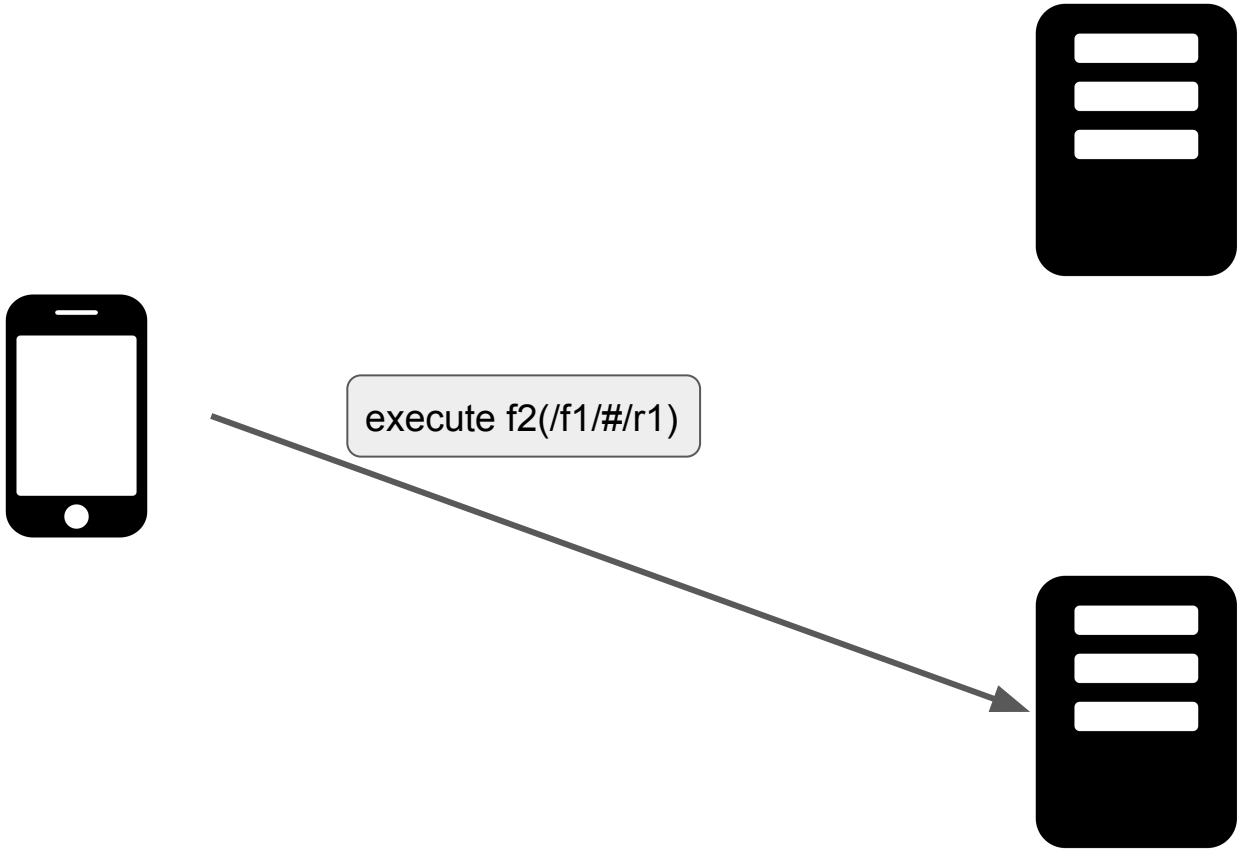
deterministic
Transparent function: /EHealth/extractFeatures/(#)/
Opaque function: /EHealth/anonimizeAudio/123/
non-deterministic

Class method: /EHealth/CoughAnalyzer/f1/(#)/
Class state: /EHealth/CoughAnalyzer/state/#/

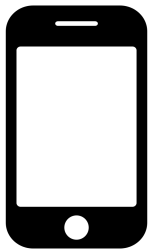
Futures

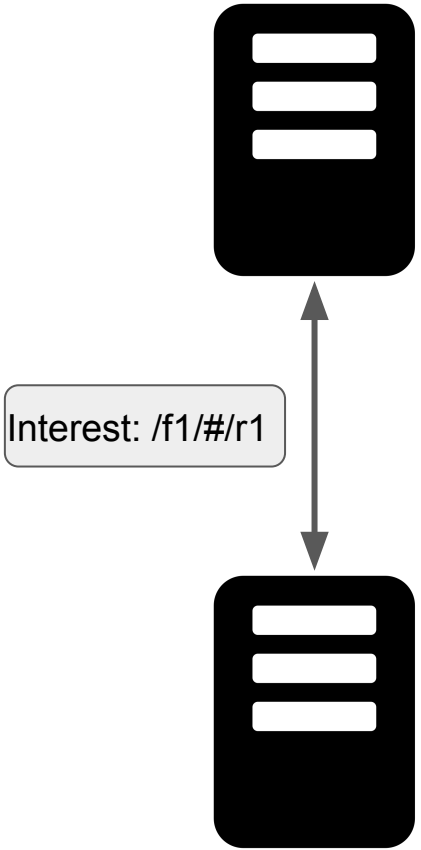


Futures



Futures





Code

Decorators:

- @cfn.transparent
- @cfn.opaque
- @cfn.actor

Methods:

- cfn.get(future)

```
class CoughAnalyzer:
    #class state
    coughs = []
    alert = False

    def addSample(self, sample_f, features_f):
        sample, features =
        coughs.append([sample, features])
        if diseaseDetected(coughs):
            alert = True


def removeSpeech(sample_f):
    sample =
    # remove speech from the sample
    return anonymized_sample


def extractFeatures(sample_f):
    sample =
    # analyze the sample
    return features
############ main ############
analyzer = CoughAnalyzer()
while True:
    sample_f = recordAudio()
    anonymized_sample_f = removeSpeech(sample_f)
    features_f = extractFeatures(anonimized_sample_f)
    analyzer.addSample(anonymized_sample_f, features_f)
```

Code

Decorators:

- @cfn.transparent
- @cfn.opaque
- @cfn.actor

Methods:

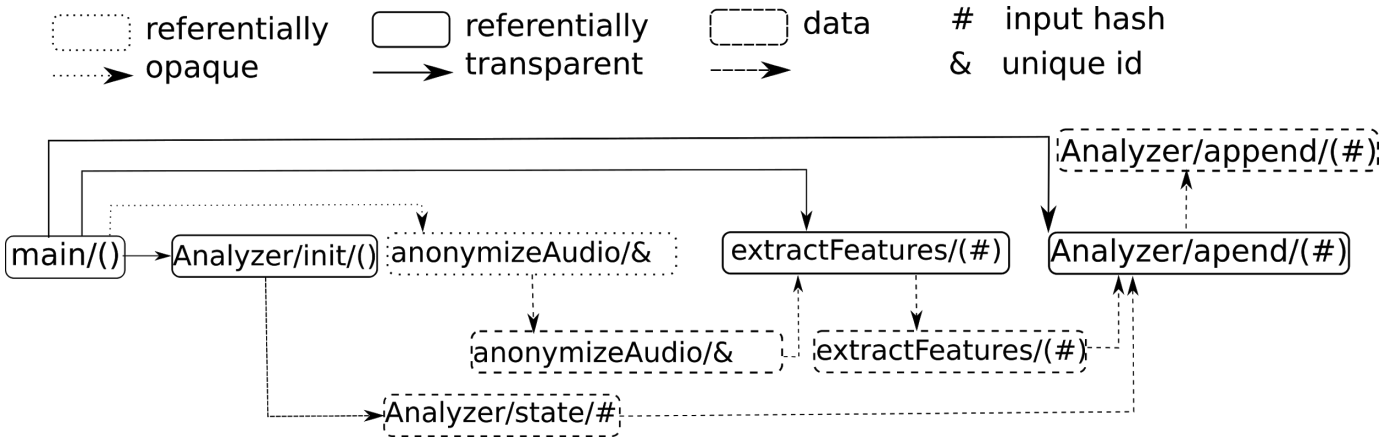
- cfn.get(future)

```
@cfn.actor
class CoughAnalyzer:
    #class state
    coughs = []
    alert = False

    @cfn.transparent
    def addSample(self, sample_f, features_f):
        sample, features = cfn.get(sample_f, features_f)
        coughs.append([sample, features])
        if diseaseDetected(coughs):
            alert = True

@cfn.opaque
def removeSpeech(sample_f):
    sample = cfn.get(sample_f)
    # remove speech from the sample
    return anonymized_sample

@cfn.transparent
def extractFeatures(sample_f):
    sample = cfn.get(sample_f)
    # analyze the sample
    return features
############ main ############
analyzer = CoughAnalyzer()
while True:
    sample_f = recordAudio()
    anonymized_sample_f = removeSpeech(sample_f)
    features_f = extractFeatures(anonimized_sample_f)
    analyzer.addSample(anonymized_sample_f, features_f)
```

Computation Graph

- Location of the data
- Chaining nodes using ICN names
- Different node types

- Graph is a CRDT
- Non-conflicting merge operations (set addition)



Computation Graph

In	**Name:** /extractFeatures/(#)	Out
/removeSpeech/(#)	**Type:** Referentially Transparent Function	/extractFeatures/(#)/r1
	Location: node1	/extractFeatures/(#)/r2
		/extractFeatures/(#)/r3

Computation Graph

In	**Name:** /extractFeatures/(#)	Out
/removeSpeech/(#)	**Type:** Referentially Transparent Function	/extractFeatures/(#)/r1
	Location: node1	/extractFeatures/(#)/r2
		/extractFeatures/(#)/r3

In	**Name:** /extractFeatures/(#)	Out
/removeSpeech/(#)	**Type:** Referentially Transparent Function	/extractFeatures/(#)/r1
	Location: node2	/extractFeatures/(#)/r2
		/extractFeatures/(#)/r3

Computation Graph

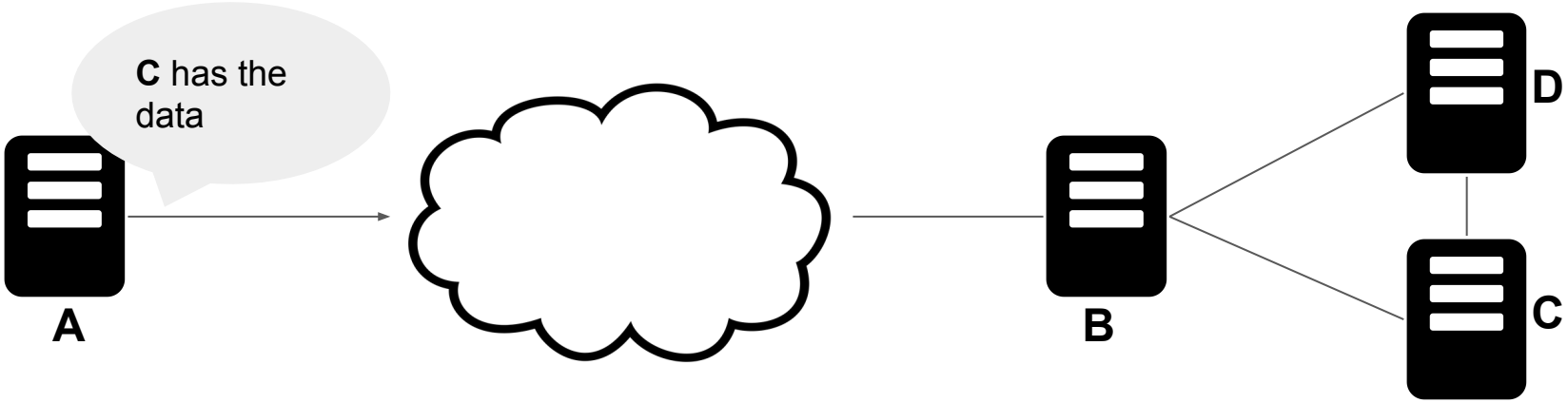
In	**Name:** /extractFeatures/(#)	Out
/removeSpeech/(#)	**Type:** Referentially Transparent Function	/extractFeatures/(#)/r1
	Location: node1, node2	/extractFeatures/(#)/r2
		/extractFeatures/(#)/r3

Task Scheduler

- Functions are invoked close to the data they rely on
- Forwarding hints to steer traffic
- Dependency information + data info are in the computation graph
- Each decision can be optimized by other forwarding nodes (late binding)
- The exact node is chosen using information from scoped resource advertisements

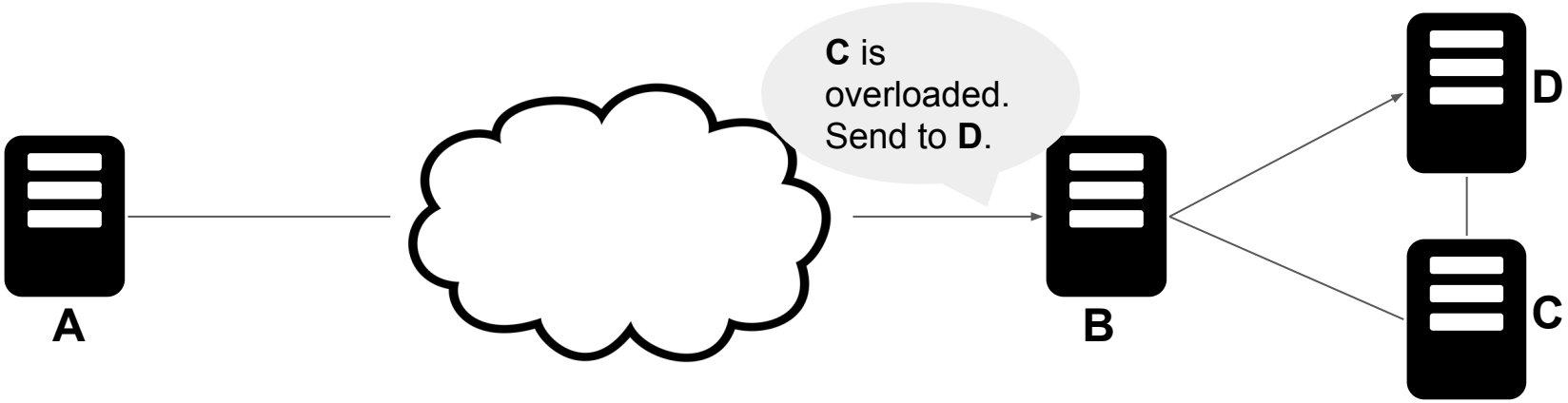
Task Scheduler

- Functions are invoked close to the data they rely on
- Forwarding hints to steer traffic
- Dependency information + data info are in the computation graph
- Each decision can be optimized by other forwarding nodes (late binding)
- The exact node is chosen using information from scoped resource advertisements



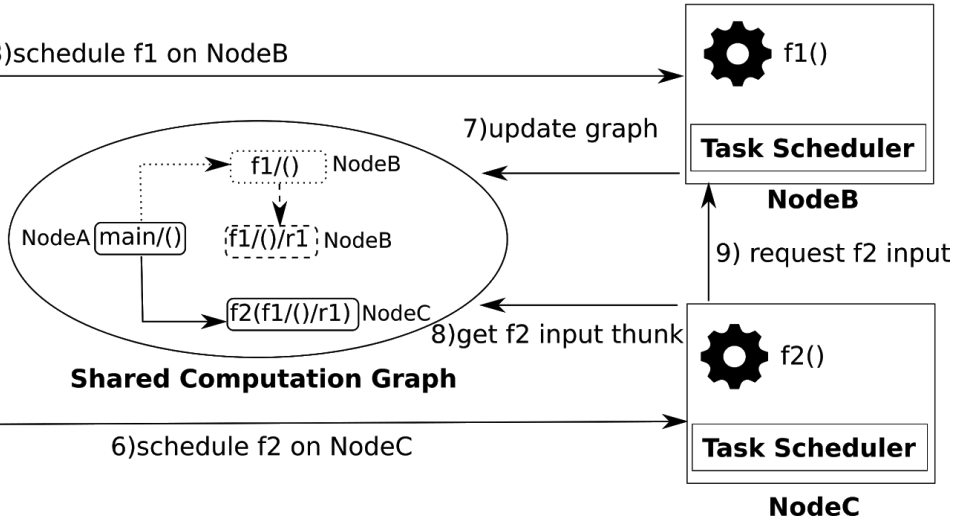
Task Scheduler

- Functions are invoked close to the data they rely on
- Forwarding hints to steer traffic
- Dependency information + data info are in the computation graph
- Each decision can be optimized by other forwarding nodes (late binding)
- The exact node is chosen using information from scoped resource advertisements



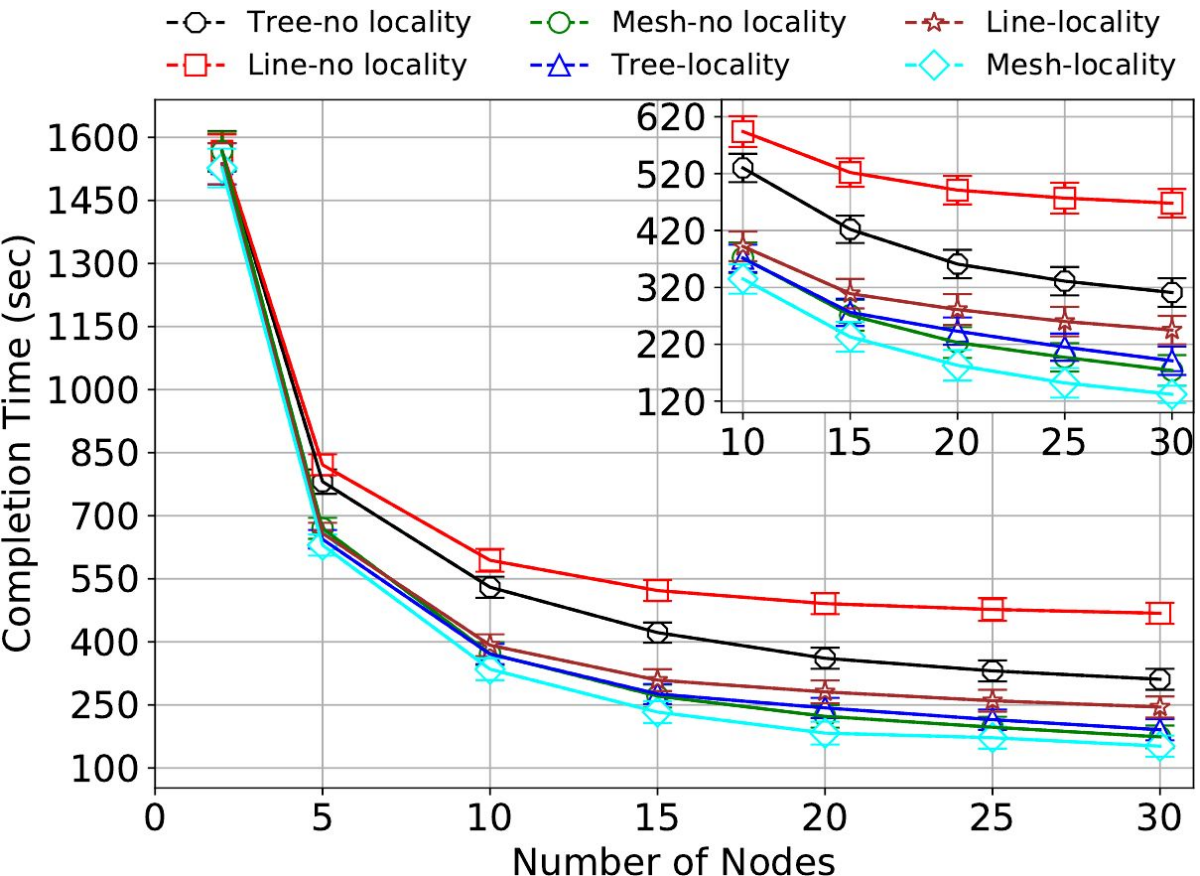
Example

```
@opaque
def f1():
    return random()

@transparent
def f2(future):
    #perform computations
    my_input = get(future)
    compute(my_input)

def main():
    f1_future = f1()

    f2(f1_future)
```



Results

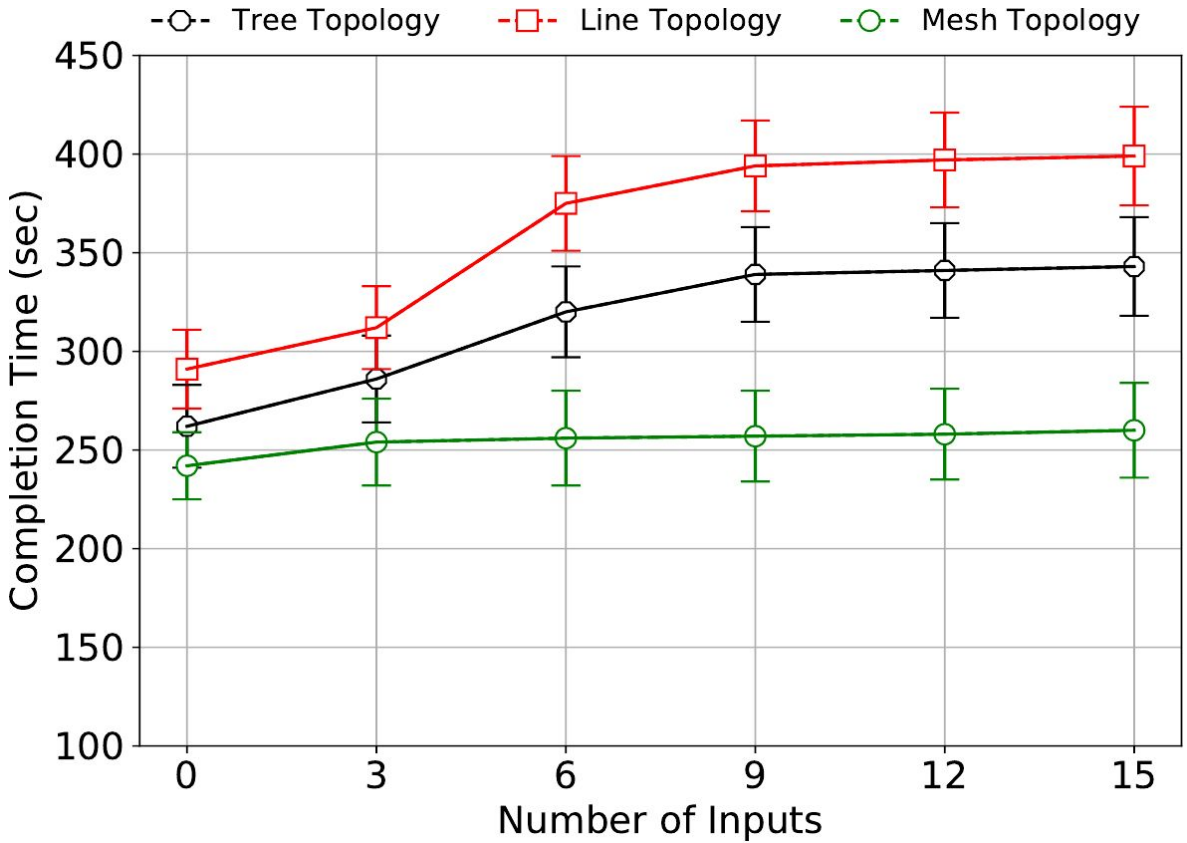
Results



- Near linear scalability
- Data locality makes a significant difference

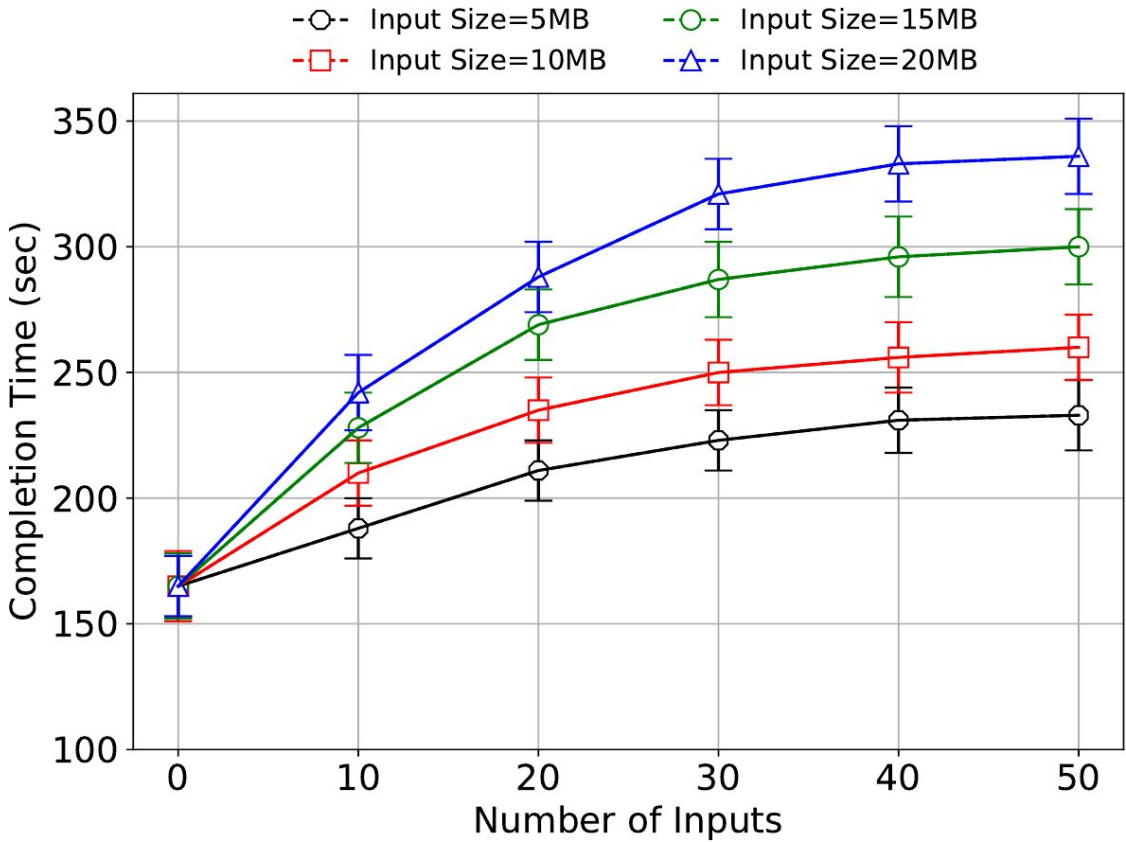
Results

- With increased number of input the completion time increases as well…
- But not that much



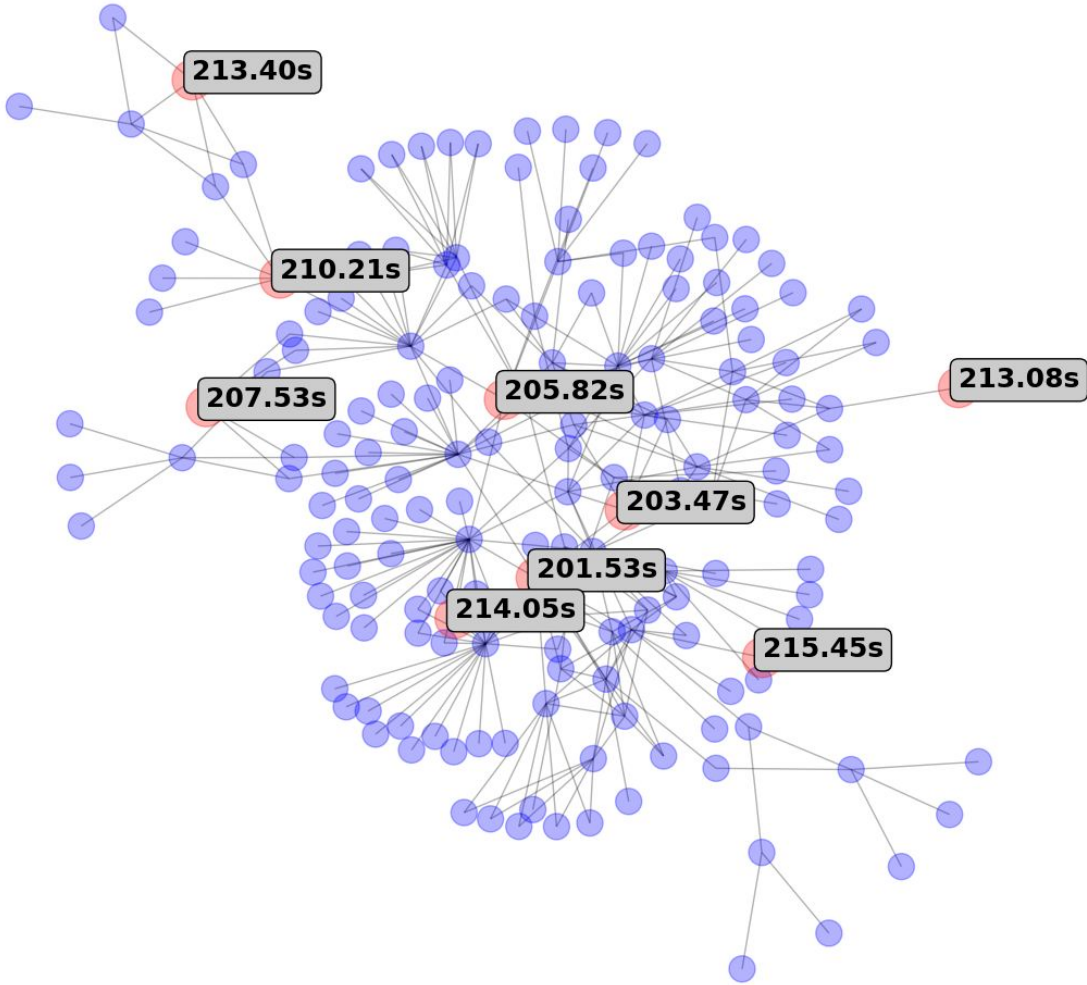
Result

- Input size plays much higher role
- The completion time is mostly determined by the largest and the furthest input



Results

- Location of the initial node does not have a big influence on the completion time



Future Work

- “Center-of-mass” approach
- Build a prototype
- Annotate real-world applications
- Automatic annotation module
- Leverage ICN mechanisms better: routing, path stitching, probing

Conclusion

- Distribute computation framework for general purpose computation
- Uses Computation Graph, Resource advertisement protocol and a scheduler
- Join optimization of network and computation resources
- Code available at https://github.com/spirosmastorakis/CFN

Thank you