Big Data from the Cloud to the Edge: The Aggregate Computing Solution

Shaukat Ali  
Simula Research Laboratory, Norway (shaukat@simula.no)  

Ferruccio Damiani  
University of Turin, Italy (ferruccio.damiani@unito.it)  

Schahram Dustdar  
TU Wien, Austria (dustdar@dsg.tuwien.ac.at)  

Marialuisa Sanseverino  
University of Turin, Italy (marialuisa.sanseverino@gmail.com)  

Mirko Viroli  
Università di Bologna, Italy (mirko.viroli@unibo.it)  

Danny Weyns  
KU Leuven, Belgium (danny.weyns@kuleuven.be)  

Linnaeus University, Sweden
Big Data sources:
IoT / mobile / edge devices
Big Data processing: edge-cloud
Ecosystem (many clouds)
Why edge-clouds?

- no global connectivity (no access to remote clouds)
- exploit locality
- exploit resources (e.g., volunteer computing)
DIES: Dependable Intelligent Edge Systems

Problem: decentralized coordination
- of edge resources and computations
- in open scenarios
- with minimal infrastructural/connectivity assumptions

Issues:
- dynamicity, failure: self capabilities
- no global connectivity: neighbour-based communication

Challenges:
- dependability
- scalability
- complexity
DIES: Dependable Intelligent Edge Systems

Problem: decentralized coordination
- of edge resources and computations
- in open scenarios
- with minimal infrastructural/connectivity assumptions

Issues:
- dynamicity, failure: self capabilities
- no global connectivity: neighbour-based communication
DIES: Dependable Intelligent Edge Systems

**Problem:** decentralized coordination
- of edge resources and computations
- in open scenarios
- with minimal infrastructural/connectivity assumptions

**Issues:**
- dynamicity, failure: **self capabilities**
- no global connectivity: **neighbour-based communication**

**Challenges:**
- dependability
- scalability
- complexity
DIES: a possible architecture

Structure:
- multiple **spatial areas (edge-clouds)**
- each area is governed by one **leader**
- agents (**workers** and **clients**)
- **relays** (agents and leaders connected through a path of relays)
DIES: a possible architecture

Structure:
- multiple **spatial areas (edge-clouds)**
- each area is governed by one **leader**
- agents (**workers** and **clients**)
- **relays** (agents and leaders connected through a path of relays)

Dynamics:
- dynamic **leader election** and **area formation**
- data/event/request **upstreaming** (agents → leaders)
- control/data **downstreaming** (leaders → agents)
DIES: envisioned architecture

Structure:
- multiple spatial areas (edge-clouds)
- each area is governed by one leader
- agents (workers and clients)
- relays (agents and leaders connected through a path of relays)

Dynamics:
- dynamic leader election and area formation
- data/event/request upstreaming (agents → leaders)
- control/data downstreaming (leaders → agents)

Desired properties:
- self-adaptive, self-organising processes
- decentralised, neighbour-to-neighbour interaction
- scalability
- dependability
What programming model?
The Aggregate Computing\textsuperscript{1} programming model

Overview:

- formally founded (field calculus)\textsuperscript{2}
- macro approach (global perspective)
- computational field (dispersed localised data as a single entity)

\textsuperscript{1} Jacob Beal, Danilo Pianini, Mirko Viroli:

\textsuperscript{2} Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, Jacob Beal:
The Aggregate Computing programming model

Structure (programming abstraction):
- devices (individual elements of an aggregate)
- neighbouring relationship (logical or physical)
- computational field (distributed data structure)
The Aggregate Computing programming model

Structure (programming abstraction):
- devices (individual elements of an aggregate)
- neighbouring relationship (logical or physical)
- computational field (distributed data structure)

Behaviour (execution protocol):
asynchronous rounds of computation

1) full run of “aggregate program” against “local context”
- device state
- sensor readings
- neighbourhood coordination data

2) broadcast coordination data to neighbours
A simple scenario

- (a) Continuous computation of a temperature threshold
- (b) Approximated by the discrete network of devices
- (c) Producing an approximation
Preliminary examples

- A program that computes whether temperature is high:
  ```
  temperature() > 20 // bool
  ```

- A program that *shows* whether temperature is high:
  ```
  if (temperature() > 20) {setLed(“green”)} {setLed(“orange”)} // unit
  ```

- A program that counts the number of rounds in each device:
  ```
  rep 0 { (x) => x + 1 }
  ```

- A program that computes whether any neighbor has a high temperature:
  ```
  anyHood (nbr { temperature() > 20 }) // bool
  ```
  // any-hood: maps each device to `whether any of its neighbor (excluding itself) has value true’
Paradigmatic example: gradient
def gradient(source) { // (bool) → num
    rep (infinity) { (d) =>
        mux (source, 0, minHood(nbr{d} + nbrRange()))
    }
}

gradient(temperature() > 20)
Abstraction and composition (1/3)

- A function that broadcasts a value from a source:

```python
def broadcast(source, value) {  // (bool, num) → num
    snd (rep (pair(infinity, value)) { (old) =>
        mux(source, pair(0, value), minHood(nbr{pair(fst(old) + 1, snd(old))))))
    }
}
```

- A function that computes the distance between a source and a destination:

```python
def distance(source, destination) {  // (bool, bool) → num
    broadcast(source, gradient(destination))
}
```
Abstraction and composition (2/3)

- A function that computes a channel:
  ```python
  def channel (source, destination, width) {
      // (bool, bool, num) → bool
      gradient(source) + gradient(destination)
      <= width + distance(source, destination)
  }
  ```

---

Ferruccio Damiani 21
Abstraction and composition (3/3)

- A program that computes a channel avoiding obstacles:
  ```python
def channel-avoiding-obstacles (o, s, d, w) {
  // (bool, bool, bool, num) → bool
  if (o) { false } { channel(s,d,w) }
}
channel-avoiding-obstacles (e\text{obstacle}, e\text{source}, e\text{destination}, e\text{width})
```
Aggregate Computing: theory and practice

Formal properties:
- Predictable composition of emergent behavior\(^1\)
- Self-stabilization\(^2\)
- Eventual-consistency\(^3\)

Proof of concept artefacts (on the Java platform):
- Toolchain and simulation framework\(^4\)
- Declarativity and flexibility in deployment\(^5\)

---

\(^1\) Giorgio Audrito, Mirko Viroli, Ferruccio Damiani, Danilo Pianini, Jacob Beal:


\(^2\) Mirko Viroli, Giorgio Audrito, Jacob Beal, Ferruccio Damiani, Danilo Pianini:


\(^3\) Jacob Beal, Mirko Viroli, Danilo Pianini, Ferruccio Damiani:


\(^4\) Mirko Viroli, Roberto Casadei, Danilo Pianini:


\(^5\) Mirko Viroli, Roberto Casadei, Danilo Pianini:

Aggregate Computing: abstraction layers

- Simple, easy to understand code
- Robust to errors, adapt to changing environment
- Scalable to potentially vast numbers of devices
- Take advantage of spatial nature of problems
Towards Implementing DIES by Aggregate Computing

Combining various design patterns:

- decentralised, self-healing gradient
- decentralised, self-healing leader election
- information flows and feedback loops
- information spreading
- information collection

---


Towards implementing DIES by Aggregate Computing

The layers involved:¹,²

¹ Roberto Casadei, Mirko Viroli:
Coordinating Computation at the Edge: a Decentralized, Self-Organizing, Spatial Approach. FMEC 2019: 60-67
² Roberto Casadei, Christos Tsigkanos, Mirko Viroli, Schahram Dustdar:
Engineering Resilient Collaborative Edge-Enabled IoT. SCC 2019: 36-45
DIES: envisioned architecture

Structure:
- multiple spatial areas (edge-clouds)
- each area is governed by one leader
- agents (workers and clients)
- relays (agents and leaders connected through a path of relays)

Dynamics:
- dynamic leader election and area formation
- data/event/request upstreaming (agents → leaders)
- control/data downstreaming (leaders → agents)

Desired properties:
- self-adaptive, self-organising processes
- decentralised, neighbour-to-neighbour interaction
- scalability
- dependability
DIES: envisioned architecture
Aggregate Computing: past, present and future

Mirko Viroli, Jacob Beal, Ferruccio Damiani, Giorgio Audrito, Roberto Casadei, Danilo Pianini: From Field-Based Coordination to Aggregate Computing. COORDINATION 2018: 252-279
Aggregate Computing: future work

Envisioned technology stack for:

- providing real-time guarantees \(^1\)
- implementing DIES

\(^1\) Giorgio Audrito, Ferruccio Damiani, Mirko Viroli, Enrico Bini: Distributed Real-Time Shortest-Paths Computations with the Field Calculus. RTSS 2018: 23-34
Thanks