Enabling distributed, compute-intensive FaaS on the edge with COMPSs

Francesc Lordan
Outline

• Motivation
• Programming Model: COMPSs
• Decentralizing the COMPSs runtime
• Use Case: Cagliari Airport
• Conclusion
Motivation
3 types of elements composing the system:

**Events/Data Generators (sensors)**
elements monitorizing certain condition and informing about it
   - Continuously generating information $\rightarrow$ Streams of data
   - Eventually notifying a change $\rightarrow$ Events

**Event-reactors/ Data Consumers (displays)**
elements on which the platform realizes actions (changing its state)

**Computing elements**
elements within the infrastructure with ability to process and transform information
Cloud-Edge Continuum
Computation on IoT/Edge platforms

3 purposes for computation:

• Sense-process-actuate:
   A sensor detects something and triggers a computation to give a proper response to such event.

• Stream processing:
   A sensor continuously provides data that needs to be processed.

• Batch Jobs:
   Analyse big amounts of data collected by the sensors
Programming Model: COMPSs
BSC vision on programming models

Simple Parallel Programming Model
- What do I need to compute?
- What data do I need to use?
- Provide Hints

Let the difficult parts to the runtime
Act on behalf of the user

Enable Monitoring and Analysis
Generate data to evaluate how application performs

Applications
Programming Model: High-level, clean, abstract

Power to the runtime
Middleware API

Cloud

Program logic independent of computing platform
General purpose Task based
Single address space
Intelligent runtime, parallelization, distribution, interoperability

What do I need to compute?
What data do I need to use?
Provide Hints

Let the difficult parts to the runtime
Act on behalf of the user

Enable Monitoring and Analysis
Generate data to evaluate how application performs
COMP Superscalar (COMPSs)

- General purpose programming language + annotations/hints
- Sequential programming with no API calls
- Agnostic of the computing infrastructure
- Task-based: task is the unit of work

- Builds a task graph at runtime that express potential concurrency
  - Implicit workflow
- Exploitation of parallelism
  - ... and of distant parallelism
public class Matmul {
    public static void main (String[] args) {
        int[][] A;
        int[][] B;
        ...
        int[][] C = multiply(A, B);
        ...
    }
}

public class Simple {
    public static int[][] multiply (int[][] A, int[][] B) {
        // Matrix multiplication code
        // C = AB
        ...
        return C;
    }
}

public interface CEI {
    @Method(declaringClass = "Simple")
    public int[][] multiply (int[][] A,
                             int[][] B)
    }
COMPSs runtime

main(args)

COMPSs Master

Application

COMPSs Runtime

Loader

C

Python

Bindings-commons

Access Processor

Scheduler

Resources

Task

Task

Task
Decentralizing the COMPSs runtime
Environment differences

COMPSs’ usual flow:

- 1 application
- Infrastructure is stable
- 1 node (MASTER):
  - Spawns tasks
  - Orchestrates the execution over the available resources (WORKERs)

COMPSs for Edge Computing

- N applications at the same time
- Infrastructure is dynamic
- Any agent may generate new tasks
- Execution orchestration is a shared responsibility
COMPSs agent

Start App Request
- Class
- Method
- Params
- CEI

COMPSs Agent

Agent Interface
Access Processor
Scheduler
Resources

Task
Application
Loader  C  Python
Bindings-commons
Agents interaction
Computing agents hierarchy
Use Case:
Cagliari (IT) airport
Smart Fog-Hub Service

In 2017 more than 4 billion passengers concentrated in airports

• Features
  • track the presence of people and objects in the field
  • proximity marketing services
  • suggestions on best use of airport services
  • Recommender system based on consumers’ behavior

• Objectives
  • Reduce latency and response times
  • Capability to distribute computing in case of overloading
  • Pleasant experience for travelers while in the airport field
Customer Experience

Flight AZ626 now boarding, gate B32

Sardinian handcraft

App install, topics setting
Android App

International food
Follow us for news about international food promotions

SMARLTVX ESTONIA 6Y08228
Destination RIGA time 02:10

ALITALIA AZ01590
Destination ROMA Fiumicino time 06:20

RYANAIR FR04707
Destination BERGAMO time 06:35

ALITALIA AZ01561
Destination MILANO LINATE time 06:45

RYANAIR FR04878
Destination ROMA CIAMPINO time 06:45

RYANAIR FR0321
Destination VERONA time 07:30

TUI TB07352
Destination METZ time 08:05

VOLOTEA V769273
Destination LIÓN time 08:05

ALITALIA AZ01588
Destination ROMA Fiumicino time 08:10

Bar Acme
Pub for breakfast and fast lunch, try our special appetizers!

International restaurant
Try our food, the best quality at the smallest price!

Discount for Japanese food in airport international restaurant
Only for the next two hours, 20% discount for Japanese food. Take advantage immediately!

ALITALIA AZ01590
Destination ROMA Fiumicino time 06:20
Gate 1 - LAST CALL
Dashboard
Use Case deployment
Benchmarking scenarios

(A) 

(B) 

(C) 

(D)
Test Results

Graph showing test results for different network conditions:
- F2C - wifi
- toCloud - wifi
- toCloud - 4G/LTE
- toCloud - 3G

Metrics include:
- msec
- min
- median
- max
Conclusions
Summary

• IoT/Edge infrastructures are composed of autonomous nodes with network and computing capabilities

• COMPSs
  • Developers code being unaware of the parallelism and infrastructure-related concerns
  • Detects tasks and the parallelism inherent in the application
  • Orchestrates the execution of these tasks on the available infrastructure
  • Supports computation on the three scenarios (batch, stream and sense-process-actuate)

• COMPSs agents
  • Allows devices to remain autonomous and compute in an isolated manner
  • By interacting with other agents, an agent has access to the available computing power

• The airport use case shows the viability and the benefits of the presented solution
THANK YOU!

francesc.lordan@bsc.es

www.bsc.es
COMPSs
COMPSs Runtime

- COMPSs applications executed in distributed mode following the master-worker paradigm
- Sequential execution starts in master node
- Tasks are offloaded to worker nodes
- All data scheduling decisions and data transfers are performed by the runtime
1. Identify tasks

main program {
    taskA(...);
    taskB(...);
}

2. Select tasks

(C++/Java)
task selection interface {
    taskA
    taskB
}

(Python)
@task decorator

Task
Unit of parallelism

Resource 1
Resource 2
Resource N
Programming Model Example

```java
for (i = 0; i < 3; i++) {
    increment(counter1);
    increment(counter2);
    increment(counter3);
}
printCounters(counter1, counter2, counter3);
```

```java
public interface SimpleItf {
    @Method(declaringClass = "SimpleImpl")
    void increment(
        @Parameter(type = FILE, direction = INOUT) String counterFile
    );
}
```

Parameter metadata

Implementation

1st iteration

2nd iteration

3rd iteration
Advanced Features

• Constraints to support heterogeneous tasks
  • @Constraints(…)

• Versioning
  • @Implements(…)

• Combination of binary execution
  • @Binary(…)

• Integration with Programming Models
  • @MPI(…) @Decaf(…)
  • COMPSs + OmpSs

• Nested
  • @COMPSs(…)
Execution Environments

- Interactive Nodes
  - Master
  - Workers
  - Task code
  - COMPSs Master RT
  - COMPSs Worker RT

- Clusters
  - Login Node
  - Queue System (LSF, PBS, ...)
  - Computing Nodes
  - Application
  - COMPSs RT

- Containers
  - Application
  - Docker Compose
  - Docker Hub
  - Docker Image
  - docker
  - runcompss_docker
  - Docker Swarm
  - Docker Nodes

- Clouds
  - Application
  - COMPSs Runtime
  - Cloud Connector
  - Provider API
  - Execute/Copy
  - Create/delete VMs
Runtime Extensions

Execution commands:
- runcompss (interactive & cloud)
- enqueue_compss (clusters)
- Runcompss_docker (socker clusters)

Runtime System

- Task Analysis
- Data Access & Locality
- Scheduling
- Job Submission & Data Transfer
- Resource Management
- Monitoring & Tracing
- Persistent Objects
- Comm. Protocols
- Resource Providers

BSC
Barcelona Supercomputing Center
Centro Nacional de Supercomputación
mF2C
Background – Open Fog

The architecture is an extension of the traditional cloud computing model

- Processes are moved from the cloud to the edge of the network, in Fog nodes
- Deployments can reside on multiple layers of a network topology,
- Deployments retain all the benefits of cloud computing, such as containerization, virtualization, orchestration, resource-efficient management

Fog Nodes peculiarities

- Autonomous processing
- Local storage and IP communications
- Hosted in open (even hostile) fields
- Capable of acting in mobility
Use case architecture

- **Cloud** – based on a OpenStack instance, wired connected with the fog layers, providing scalable computing power for Machine Learning algorithms

- **Edge Fog** – with a fog aggregator based on Nuvlabox with 8GB, providing real-time computing and storage resources and 6 rPi with 1GB hub providing session management and fast response to the edge devices

- **Edge IoT** – Android smartphones connected to the edge nodes through wifi, and using an Android app to interact with the system
Testbed

• Proximity processing
  • Client – call a request for a list of nearby POIs using geographic position
  • Server – calculates the POIs in proximity and returns a JSON array

• Client
  Smartphone XIAOMI with Android 5.0.2, running app calling a Rest HTTP API

• Server
  • A VM runs a dockerized image with proximity calculation
    1. rPI3 with 1GB RAM
    2. VM running on a public cloud (4-core processor and 4GB RAM)
Interaction with IoT devices
Interaction with sensors

• Every sensor is attached to one device with computing capabilities

  • Events:
    the device trigger a function execution on any COMPSs agent

  • Streams:
    the device publishes the stream of data directly
Interaction with displays/devices

- Through a Controller class
  - Handles all the communication with the device
  - Offers a simple API to interact with the device
- Each instance of this controller class interacts with one device
- Device can be controlled by any agent if the object is transferred

```java
public void task(Streetlamp sl) {
    ...
    sl.on();
    ...
}
```

```java
class Streetlamp {
    public void on() {
        ...
    }
    public void off() {
        ...
    }
}
```
Interaction with displays/devices

- Using the storage framework
  - Controller class is a Persistent Object
  - One Controller instance is made persistent for each device using a universal ID
  - Any agent able can interact with any registered resource by using an ID

```java
public void task() {
    ... 
    Streetlamp sl;
    sl = Storage.getByAlias("light21758");
    sl.on();
    ...
}
```

```java
class Streetlamp extends StorageObject {
    public void on() {
        ...
    }
    public void off() {
        ...
    }
}
```