TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

Macroarea di Ingegneria
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Projects

Corso di Sistemi Distribuiti e Cloud Computing

A.A. 2023/24

Valeria Cardellini

Laurea Magistrale in Ingegneria Informatica

Choice and deadline

- Project choice is mandatory
- Deadline: February 15, 2024
- How to choose: send me an email with the following info
  - Email subject: SDCC prenotazione progetto
  - Team members (for each student: name, email and student ID number)
  - Chosen project (and description of service, if applicable)
- Maximum number of available slots for each project
  - Assigned with FIFO discipline: after my presentation, not during! ☺
- Communicate promptly and motivate any change to your team
- Project is valid only for A.Y. 2023/24

Delivery: what

- What to deliver
  - Link to shared cloud folder or code repository containing:
    - Code
    - Instructions to configure and run your code
    - Report
    - (if applicable) Dataset of experimental results
  - Write your report using the structure of as scientific article
    - E.g.,: Title, Author(s), Abstract, Introduction, Background, Solution Design, Solution Details, Results, Discussion, References
    - Maximum 8 pages using ACM or IEEE double-column format
      - ACM proceedings templates https://www.acm.org/publications/proceedings-template
      - IEEE proceedings templates https://www.ieee.org/conferences/publishing/templates.html

Delivery: when

- When to deliver project code and report
  - By September 20, 2024
  - About one week before project presentation
  - No prefixed dates for presentation, we will agree on the date after you deliver the project
  - Team members > 1: All team members discuss their project on the same day

Presentation

- What to present
  - Prepare slides
  - Prepare live demo of project
  - Team members > 1: Each team member discusses a part of the project (it is your choice which part)
  - Maximum 10 minutes per member
    - I will check time and interrupt you if needed
    - Live demo is not included!
  - Q&A during and at the end of presentation

Common requirements for all projects

- Programming language: depends on project
- You can use support libraries and tools to develop your project (of course they should not overlap with the project goals!)
  - Be careful: their use must be properly mentioned in the project report
- System/service with configurable parameters (no hard-coded!)
  - Through a configuration file/service
- You must test all the functionalities of your developed system/service and present and discuss the testing results in the project report

Common requirements for all projects (2)

- System/service state should be distributed
  - The only allowed centralized services can be one to provide service discovery, users logging, and other housekeeping tasks
- System/service supports multiple entities (e.g., concurrent clients) which may contend for shared resources
- System/service supports update to some form of shared state
- Depending on chosen project:
  - System/service scalability and elasticity
  - System/service fault tolerance
    - In particular, system/service continues operation even if one of the nodes crashes (optional: a crashed node recovers after crash so that it can resume operation)

Grant for cloud services

- Projects require the use of Amazon Web Services (AWS) through Learner Lab provided by AWS Academy
  - You should have received the email to access the grant, let me know if you cannot find it
  - 100 $ grant
  - Some limits on services, check the list of available services!
- Plus AWS Free Tier for 12 months (unless you have already registered for AWS account)

Project types

- See our first lesson of the course
- Type A
  - Final score:
    - 50% written exam (plus elective oral exam)
    - 50% project (2-3 students per team)
- Type B
  - Final score:
    - 75% written exam (plus elective oral exam)
    - 25% individual project

- Change of project type (from A to B or viceversa) is not allowed

Projects A: summary

- A1: System for task processing in the cloud-edge continuum
  - 2-3 students per team, max 3 teams
- A2: Microservice app for green smart cities
  - 2-3 students per team, max 3 teams
- A3: Energy/carbon-aware scheduling of serverless workflows
  - 2-3 students per team, max 2 teams
- A4 (*joint with ML course*): Learning-aware caching of containers for serverless workloads
  - 2-3 students per team, max 2 teams
- For all A projects: programming language of your choice

Project A1

- System for task processing in the cloud-edge continuum
  - 2-3 students per team, max 3 teams
  - Resource constrained devices at edge, powerful resources in cloud
  - Select in a smart way the edge node for task processing (network proximity, load, …)
  - Process low-demanding and latency-sensitive tasks at edge nodes
  - Dynamically offload high-demanding computing tasks to another edge node or to cloud
  - Optional: migrate a running task to another node

Project A2

- Microservice app for green smart cities
  - 2-3 students per team, max 3 teams
  - Application related to the Green Revolution and Ecological Transition PNRR theme
    - Sustainable agriculture and circular economy
    - Waste management
    - Energy consumption
    - Health and telemedicine (e.g., remote patient monitoring)
  - Programming language of your choice
  - Requirement: well-defined and documented API
  - One team will be selected to participate to CINI Smart Cities University Challenge, co-located with I-Cities 2024
  - Deadline: July 15 (by July 31: online meeting with Challenge organizers and teams from other universities)
  - Option: can be extended as joint project with ML course

Project A3

- Energy/carbon-aware scheduling of serverless workflows
  - 2-3 students per team, max 2 teams
  - Multiple geographical regions in which serverless functions can be executed
  - Decide where to schedule functions
  - Take into account carbon footprint of electricity at different regions at the time of scheduling and/or energy consumption on computing nodes
  - Can be implemented to schedule serverless functions on AWS Lambda or an open-source FaaS framework (e.g., OpenWhisk, OpenFaaS, Knative)

Project A4

- Joint project with ML course
- Learning-aware caching of containers for serverless workloads
  - 2-3 students per team, max 2 teams
  - Design and evaluate a ML-based caching policy to reduce cold starts of serverless workloads
  - Realize a basic system to run serverless functions in containers or implement your caching policy into our Serverledge system https://github.com/grussorusso/serverledge

Projects B: summary

- B1: Replication transparency for Go RPCs
- B2: Distributed leader election
- B3: Chandy-Lamport distributed algorithm
- B4: Gossip-based distance estimation and failure detection
- For all B projects:
  - Go as programming language
  - 1 student per team, max 5 students per project

Project B1

- Replication transparency for Go RPCs
  - 1 student per team, max 5 students
  - Programming language: Go
  - Extend your Go exercise
  - Manage service discovery (do not use a SQL database!)
  - Manage server crashes
  - Design and evaluate a state-aware balancing policy (e.g., randomized load-aware)
  - Design and evaluate a mechanism to reduce tail latency (invoke RPC on two servers, take the first response and stop processing on late server)
  - Implement different RPC tasks (including a heavy one)
  - Test your solution (including node crash)
  - Deployment using Docker Compose and EC2 instance

Project B2

- Distributed leader election
  - 1 student per team, max 5 students
  - Programming language: Go
  - Implement 2 distributed leader election algorithms (one can be either bully or ring election, the other from literature)
  - Manage service discovery (do not use a SQL database!)
  - Manage server crashes
  - Test your solution (including node crash)
  - Deployment using Docker Compose and EC2 instance

Project B3

- Chandy-Lamport distributed algorithm
  - 1 student per team, max 5 students
  - Programming language: Go
  - Famous algorithm to record a global consistent snapshot of a distributed system/application
    - A snapshot records the local state of each process along with the state of each communication channel used by the processes to communicate
  - Test your solution on a pipeline distributed application
  - Deployment using Docker Compose and EC2 instance

Project B4

- Gossip-based distance estimation and failure detection
  - 1 student per team, max 5 students
  - Programming language: Go
  - Implement gossip-based algorithms to estimate node distance (e.g., Vivaldi) and detect failures
    - Vivaldi is the most widely used decentralized NC system, which assigns coordinates to nodes in a Euclidean space such that the distance between two nodes accurately predicts the network latency between the nodes
    - Using a network coordinate (NC) system, no need to perform direct latency measurements between each pair of nodes!
    - Other network coordinate systems exist, e.g., Pharos
  - Test your solution (including node crash)
  - Deployment using Docker Compose and EC2 instance
  - You also need to emulate network delays (e.g., netem tool)