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 B5

- Simple distributed key-value store with consistency guarantees
  - 1 student per team, max 4 students
  - Programming language: Go
  - Implement a simple distributed key-value store with consistency guarantees
    - Operations: get(key), put(key, value), delete(key)
    - Clients can request operations to any replica
    - Provide consistency guarantees
    - Sequential consistency using logical scalar clocks
    - Causal consistency using logical vector clocks
  - Test your solution (at least 3 replicas)
  - Deployment using Docker Compose and EC2 instance
  - You also need to emulate network delays to simulate delays in the message transmission among different replicas