# Distributed Software Systems Assignment 2


- Riccardo Benedetti
- Leon Harz

# What is a scalable system?

- A scalable system is a system  which is able to handle an increasing amount of work, by adding resources to a system.
- In particular, there are two ways to “scale” a system
  - Vertical scaling: Increasing the capacity of a single machine (e.g., upgrading the hardware like CPU, RAM).
  - Horizontal scaling: Adding more machines to share the workload (e.g., adding more servers to a network).

# Some examples….

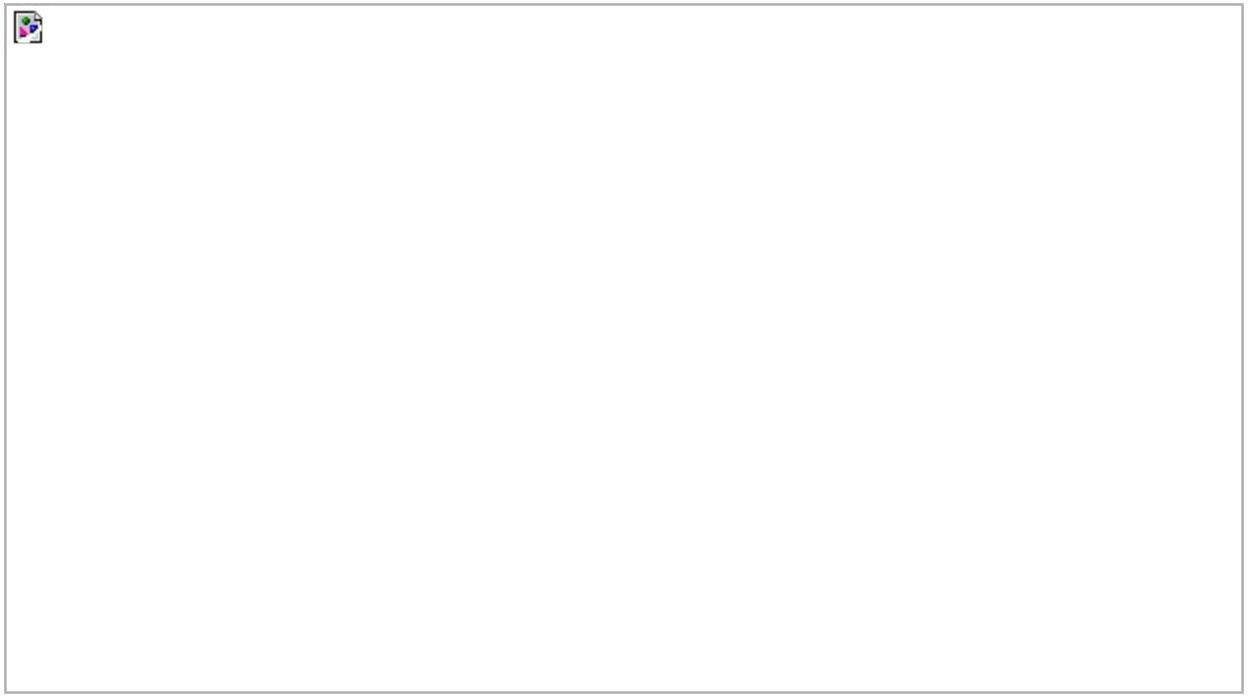
- Cloud Computing Services: these services adapt to the workload by automatically adjusting resources, using the “elasticity” property of a distributed system.
- Databases with Replication functionality: they can scale to handle more read/write operations, without overloading a single server
- CDNs (Content Delivery Networks): they use an array of multiple servers distributed among different geographical locations to serve contents efficiently.

Note: Model used for this slide GPT4-o. Prompt (“Give me some real world examples of scalable systems”)

How might the clocks in two computers that are linked by a local network be synchronized without reference to an external time source?

- To synchronize two computers linked by a local network we use “Cristian’s Protocol”.
- To implement it, a computer, which we call “A”, acts as a “Time Server”, a second computer “B”, requests time from “A”, when “A” receives the request, it send the current clock to “B”
- When “B” receives the time from “A” it estimates the time based on the Round Trip Time for the message exchange. “B” adjusts its clock by setting the time according to Time + RTT/2, this is done to compensate for the time it took the package to travel from “A” to “B”.

# Cristian’s Algortim Visualization



What factors limit the accuracy of the procedure you have described?

- Network Congestion: Depending on the traffic on the local network, the Round Trip Time may vary, leading to inaccurate clock adjustments.
- Processing Delays: Both machines (A and B) need to time to process and elaborate the responses and requests, potentially losing significative accuracy in the clock synchronization.
- Message Loss: If a message is lost, it needs to be retransmitted, adding further delay.

How could the clocks in a large number of computers connected by the Internet be synchronized?

- Adding a “Switch” computer, which acts as a global time server, which carries Christian’s Protocol, along with all the other node machines connected to it.
- The accuracy of this procedure depends on the frequency and regularity of the synchronization checks, as discussed in the previous slide.
- Clock Drift: is a gradual deviation of a computer's internal clock from the correct time. It happens because the hardware clocks in computers are not perfectly accurate. A system should be implemented to correct and prevent clock drift, like assuring the nodes remain connected to a stable internet connection.

# Exercise 2: Simple File Server

- Task:
  - Build a server and client application, that enables the user to read a file from the server or locally without him/her knowing where the file is located
  - Ensure network transparency

- Used Tools:
  - We decided to use REST API via Flask for Python
  - REST-API communicates over http-requests
  - Flask abstracts a lot of low level network communications

# Technical Overview Server

- Server:
  - Server creates a storage directory if not present
  - If storage + files were present, server reads index.json to know which files are stored
- Receive:
  - When a client wants to upload a file the server stores the file in the storage directory
  - Updates index file
  - Responds to the client accordingly (file received + stored-> success otherwise error)
- Send:
  - When a client wants to access a file stored on the server, the server checks if file is present
  - If present server loads the content of the file and sends back to the client
  - Otherwise response with error code

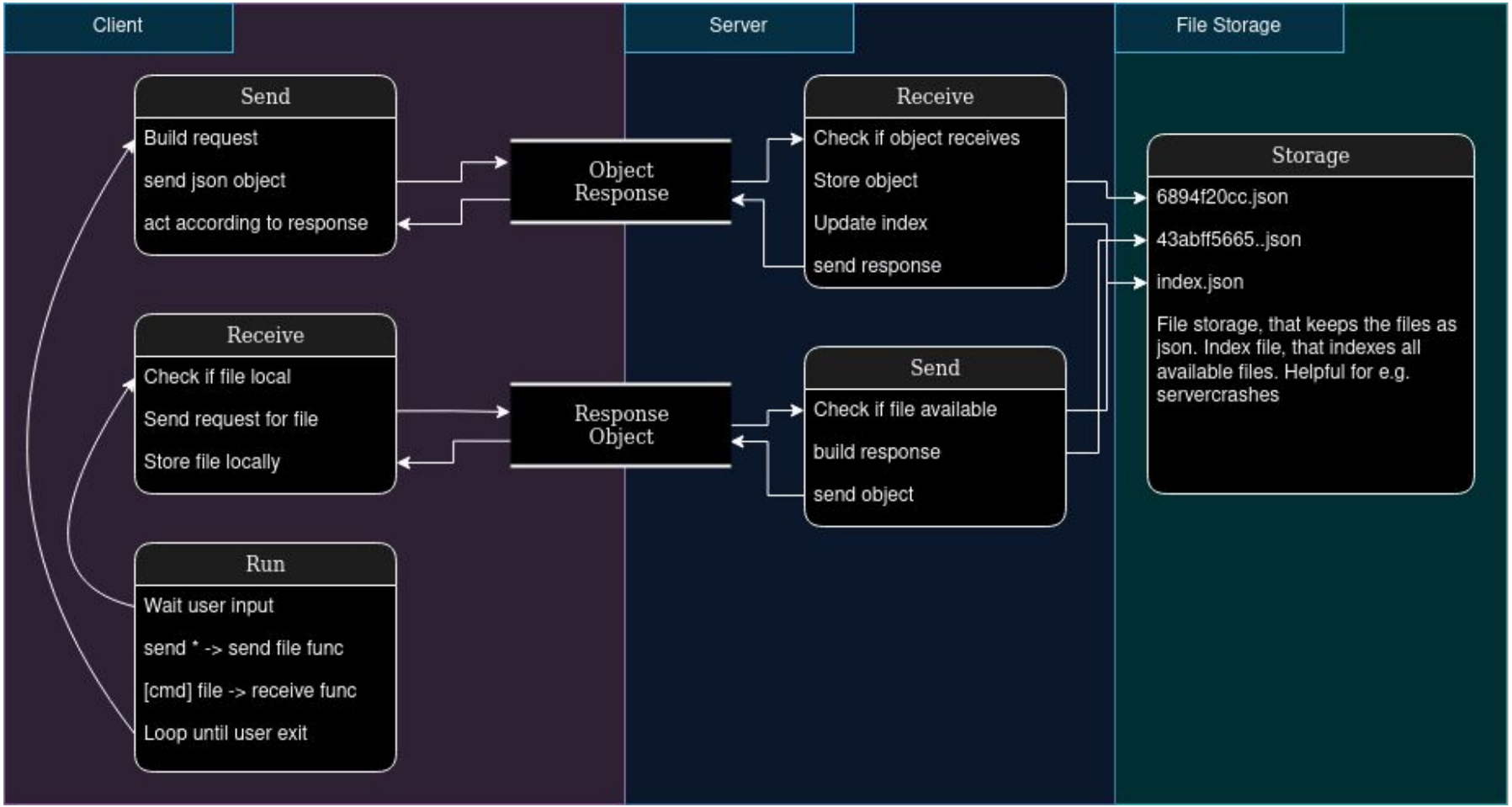
# Technical Overview Client

- Client:
  - Creates a client storage if not present (crucial)
  - Runs in a while loop, while waiting for user commands
  - parses user commands and checks if input valid
- Send:
  - Creates unique URI from file+path
  - Sends file to Server
- Receive:
  - Checks if file is stored locally (client storage)
  - Creates unique URI from file+path
  - Sends request of that URI to server
  - File is present: Client executes the given command with file

# Server-Client Communication

- Scenario 1 (User wants to store a file on the server):
  - User specifies the file that should be send
  - Client creates a unique hash out of filename + path
  - Client sends the content of file + additional file informations to the server
  - Server sends response to client (either positive or negative)
- Scenario 2 (User wants to read a file):
  - User provides a simple command as the desired file
  - Client checks if file is stored locally
    - if stored locally command is executed
  - Client calculates the unique hash of file
  - Client sends a request to the server asking if the file is present
    - if not server responds accordingly
  - Server sends file back to client
  - Client executes the provided command

# Graphical Overview



# Network Transparency

- Network transparency is provided by the client
  - Client checks if file is stored locally
  - Client requests file if not stored locally
  - User has the same experience independent on file location

- The client handles all the server communication
  - successful responses never reach the user directly
  - only in cases where the file is not stored on both resources user gets server responses