Politecnico di Milano
Middleware Technologies for Distributed Systems
Prof. A. Margara and G. Cugola

Projects for the academic year 2019-2020

Rules

1. Projects must be developed in groups composed of a minimum of two and a maximum of three students.
2. The set of projects described below are valid for this academic year, only. This means that they have to be presented before the last official exam session of this academic year.
3. Students are expected to demonstrate their projects work in a real distributed system
    - Two or more laptops (or virtual machines).
    - Cloud deployment (we can provide Microsoft Azure credits).
4. Students are expected to present the software architecture and the algorithms / protocols implemented in the projects. They can use a few slides to support their presentation.
5. Each student in a group is expected to answer questions on any part of any project developed by the group.
6. Each group can do either the Apache Spark project or the Apache Flink project, but not both.
7. Students interested in doing their thesis with either lecturer can skip the project corresponding to their thesis topic.

Distributed key-value store in Akka

Implement a distributed key-value store in Akka

Requirements

- Clients can interact with the store using two primitives
    - put(K, V) inserts/updates value V for key K.
    - get(K) returns the value assigned to key K.
- The store should support scaling by partitioning the key-space and by assigning different keys to different nodes.
- The store should store each data element into R replicas to tolerate up to R-1 simultaneous failures without losing any information.
    - R is a configuration parameter.
    - Upon the failure of a node, the data it stored is replicated to a new node to ensure that the system has again R copies.
- To avoid write-write conflicts, all writes are performed on a leader replica that decides their order.
- To improve availability, clients can read from any replica.
    - Optional: the store ensures client-centric consistency.
- New nodes can be added to the store dynamically.

Assumptions

- Processes can fail, while channels are reliable.
- You can assume a master / supervisor node that never fails.
- You are allowed to use the Akka clustering / membership service.

Processing pipeline in Kafka

Implement a data processing pipeline in Kafka.

Requirements

- Provide administrative tools / scripts to create and deploy a processing pipeline that processes messages from a given topic.
- A processing pipeline consists of multiple stages, each of them processing an input message at a time and producing one output message for the downstream stage.
- Different processing stages could run on different processes for scalability.
- Messages have a key, and the processing of messages with different keys is independent.
    - Stages are stateful and their state is partitioned by key (where is the state stored?).
    - Each stage consists of multiple processes that handle messages with different keys in parallel.
- Messages having the same key are processed in FIFO order with end-to-end exactly once delivery semantics.

Assumptions

- Processes can fail.
- Kafka topics with replication factor > 1 can be considered reliable.
- You are only allowed to use Kafka Producers and Consumers API
    - You cannot use Kafka Processors or Streams API, but you can take inspiration from their model.
- You can assume a set of predefined functions to implement stages, and you can refer to them by name in the scripts that create and deploy a processing pipeline.

Processing of car accidents data

The goal of this project is to infer qualitative data regarding the car accidents in New York City.

The data set available on Beep. Each row in the data set contains the following data:
DATE, TIME, BOROUGH, ZIP CODE, LATITUDE, LONGITUDE, LOCATION, ON STREET NAME, CROSS STREET NAME, OFF STREET NAME, NUMBER OF PERSONS INJURED, NUMBER OF PERSONS KILLED, NUMBER OF PEDESTRIANS INJURED, NUMBER OF PEDESTRIANS KILLED, NUMBER OF CYCLIST INJURED, NUMBER OF CYCLIST KILLED, NUMBER OF MOTORIST INJURED, NUMBER OF MOTORIST KILLED, CONTRIBUTING FACTOR VEHICLE 1, CONTRIBUTING FACTOR VEHICLE 2, CONTRIBUTING FACTOR VEHICLE 3, CONTRIBUTING FACTOR VEHICLE 4, CONTRIBUTING FACTOR VEHICLE 5, UNIQUE KEY, VEHICLE TYPE CODE 1, VEHICLE TYPE CODE 2, VEHICLE TYPE CODE 3, VEHICLE TYPE CODE 4, VEHICLE TYPE CODE 5

By contributing factor we intend the reason why the accident took place in the first place.

Extract the following information

- Number of lethal accidents per week throughout the entire dataset.
- Number of accidents and percentage of number of deaths per contributing factor in the dataset.
    - I.e., for each contributing factor, we want to know how many accidents were due to that contributing factor and what percentage of these accidents were also lethal.
- Number of accidents and average number of lethal accidents per week per borough.
    - I.e., for each borough, we want to know how many accidents there were in that borough each week, as well as the average number of lethal accidents that the borough had per week.

OpenMP and MPI

- Your code should exploit both OpenMP within a single process and MPI to distribute the processing load across multiple machines.
- Provide an analysis of scalability by evaluating the performance when increasing the number of OpenMP threads and/or the number of MPI processes.
    - To increase the cost of processing, you can artificially augment the dataset by repeating each entry multiple times.
    - In your analysis, consider how different phases (e.g., loading data from file, data processing, communication, output of results) contribute to the overall execution time.

Apache Spark and Apache Flink

- Your code should run in cluster mode, with at least two slaves (Apache Spark) / TaskManagers (Apache Flink) running on different physical or virtual machines.
- You can write your code either using functional (core) API or using higher level API (Spark SQL / Flink Table API).
- Provide an analysis of scalability by evaluating the performance when increasing the number of slaves and resources (cores) provided to slaves.
    - To increase the cost of processing, you can artificially augment the dataset by repeating each entry multiple times.
    - In your analysis, consider how different phases (e.g., loading data from file, data processing, communication, output of results) contribute to the overall execution time.

In-network data collection and processing with TinyOS

Implement a reactive data collection infrastructure that informs a sink node when the temperature measured by any sensor goes above a given threshold. The sink node periodically chooses the new threshold and broadcasts a SETUP message, which floods the network (assume a multi-hop scenario).

The SETUP message is used to propagate the new threshold and to setup the routes that must be followed back by DATA messages to reach the sink.

Each sensor periodically measures the temperature and delivers a DATA message toward the sink if the measured value is above the current threshold. DATA messages are sent hop-by-hop toward the sink using point-to-point communication (use the address-based protocol provided by the TinyOS active message layer). The DATA message carries the measured temperature and the identifier of the measuring node.

Assumptions

- Assume DATA messages are rare (i.e., the threshold is high with respect to the average temperature).

Requirements

- Test the system in TOSSIM, with a network big enough to stress the multi-hop nature of the protocol.

Image server with REST API

The goal of this project is to create an image server that provides a REST API for

- Creating new users (each user should have a unique id and a name).
- Listing the users in the system.
- Uploading .jpg images to a user's online storage space (each image should have a unique key and a title).
- Listing the available images by user id.
- Obtaining the images from the server.

Requirements

- The API should be developed in such a way as to support Hypermedia.
- Optional: the image server's REST API supports OAuth to allow third party applications to access a user's images.