Κατανεμημένα Συστήματα 2016

1η Φάση εργασίας

Ομάδα 35
3120005 - Αλιπράντης Ευστάθιος
3110089 - Κοτρώτσιου Σταυρούλα
3120176 - Στεργίου Νικόλαος

Project code is located at github.com/staurou/DistributedTopCheckins

Report

The system consists of 4 programs:

- mapper - Handles the map operation
- reducer - Handles the reduce operation
- master - Responsible for accepting client request, forwarding them to the mappers, receiving the response from the reducer and sending the response to the client
- dummyclient - Desktop app that provides a UI for sending requests to the system and viewing the results.

Running master

From the dist directory run
`java -jar Katanemimena.jar master <options>`
Where options are in the following format:
[-p CLIENT_PORT] [-cp CONTROL_PORT] -m MAPPER_ADDRESS MAPPER_PORT [MAPPER_ADDRESS MAPPER_PORT]... [-r REDUCER_ADDRESS [REDUCER_PORT]]

- -p CLIENT_PORT defines the port to listen for client requests. Defaults to 25697
- -cp CONTROL_PORT defines the port to listen for internal control requests. Defaults to 25698
- -m MAPPER_ADDRESS MAPPER_PORT... is a sequence of host-port pairs declaring the address of the mappers. Defaults to an empty list.
- -r REDUCER_ADDRESS [REDUCER_PORT] declares the address of the reducer. Defaults to localhost:25700

NOTE that master does not start any other program. The mappers and the reducer must be started with a separate command. This can be done prior or after starting the master.

Basic example
`java -jar Katanemimena.jar master -m localhost 5894  83.212.117.76 5895`
Runs master with default client and control ports, declaring the reducer at default localhost:25700 and two declared mappers: one at localhost:5894 and one at 83.212.117.76:5895.

Running mapper

From the dist directory run

`java -jar Katanemimena.jar mapper <options>`

Where options are in the following format:

[-p PORT] [-r REDUCER_ADDRESS [REDUCER_PORT]] [-s MASTER_ADDRESS]

- -p PORT defines the port to listen for requests. Defaults to 25701
- -s MASTER_ADDRESS declares the address of the master. Defaults to localhost:25698
- -r REDUCER_ADDRESS [REDUCER_PORT] declares the address of the reducer. Defaults to localhost:25700

Running reducer

From the dist directory run

`java -jar Katanemimena.jar reducer <options>`

Where options are in the following format:

[-p PORT] [-m MAPPER_ADDRESS [MAPPER_ADDRESS] [-s MASTER_ADDRESS [MASTER_CONTROL_PORT]

- -p PORT defines the port to listen for mappers.Defaults to 25700
- -m MAPPER_ADDRESS declares the address of the mappers. Defaults to an empty list.
- -s MASTER_ADDRESS [MASTER_CONTROL_PORT] declares the address of the master.Defaults to localhost:25698

Running dummyclient

From the dist directory run

`java -jar Katanemimena.jar dummyclient`

Communication between programs

Programs communicate using TCP/IP sockets at configurable ports.

Each program listens for requests on a specific port and accepts connections only from known hosts. For example, reducer will accept connections only from mappers, master, localhost and 127.0.0.1

The master thought uses a different port for internal connections between the systems's programs and for client connections. A client connection to master may originate from any host.

The data exchange format is textual JSON.

Threading and Parallelism

All programs are using asynchronous, event-driven, non-blocking IO.

When an IO event occurs, such as a new request connection or new data being read, the *result* is dispatched to a thread from the pool for processing. This means that no threads are blocking waiting for connections or new data, thus, leaving resources available while IO operations are being performed.
The number of threads processing the results of IO operations is configured to be equal to the number of available processors. Meaning, that while an arbitrary number of requests can be submitted simultaneously, the number of requests being actively processed at a time (leaving out IO operations) equals the number of processors.

Client Request Management

Each client request that arrives on master is given a unique request id.
This request id is stored by the master along with the initial client connection while the request is being processed. When the request is done processing or failed, the initial connection with the client is retrieved by the request id; the result (or errors messages) are sent via that connection, the connection gets closed and the request id is deleted from the system.

Additionally, the reducer starts the reducing process when all mapper results associated with a specific request id have arrived.

Format of the Client Request

The client request consists of a geographical area (latitude and longitude range) and a time interval.

Distributing the Request to Mappers

The function that breaks the original client request to pieces and distributes it to the mappers must satisfy the following criteria:

1. Checkins from the same POI must be handled by the same mapper because the mapper's job is to count the number of checkins of each POI so the mapper should be able to access all the checkins of a POI that he processes.
2. Checkins must be as equally distributed to mappers as possible.

The first criterion implies that geographical properties should be taken into account when breaking the request to pieces and distributing checkins to mappers.
The second criterion implies that if the original request is broken to smaller geographical areas and each sub-area is assigned to a mapper, these sub-areas must be small and interlaced to eliminate the case where a mapper is assigned a highly populated area and undertakes a great part of the load while another mapper is assigned sparse areas.
But these smaller geographical areas must also not be extremely small because the location of two checkins of the same POI may be slightly different.

So the proposed function is the following:
Mapper number **`i`** handles checkin at (**`long, lat`**) if

$$| (FLOOR(long * 100) + FLOOR(lat * 100)) \ MODULO \ mappersCount | = i$$

Mapping Operations

The input of the mapper is a set of geographical areas, a time interval and a number k.
The output must be a mapping of POIs to number of checkins per POI, without counting checkins with the same photo. Only top-k POIs must be included in the output, which is pushed to reducer.

The mapper executes the following procedure:

1. Request from the data source all checkins from the assigned geographical areas which happened in the specified time interval ordered by POI and photos URL.
2. Foreach checkin sequentially:
   1. If current checkin has the same POI and photos URL as the previous one, ignore it as duplicate.
   2. If current checkin has the same POI as the previous one, increase the checkin count of that POI by one.
   Else, add the previous POI to the `ListOfTopPois` only if the previous POI is currently one of top k POIs in terms of checkins count, limiting the size of the `ListOfTopPois` to k.
3. Push the `ListOfTopPois` to reducer.

Reducing Operations

The input of the reducer is a set of responses (POIs and numbers of checkings per POI) coming from the mappers.
The output must be a mapping of the request id and the top-k POIs in the given geographical area.

The reducer starts the process when all mappers finish operating on a specific request and executes the following procedure:

1. Combines the results of the mappers into a single list.
2. Sorts that list.
3. Pushes the top-k elements of the list to master.