ADCC

Progetti anno 2024/2025

Progetti

- Sono disponibili 2 progetti
  - Progetto facile (max 24/30) Implementazione di un spazio di Tuple
  - Progetto difficile (max 30L) Implementazione di una DHT basata su Kademlia

- La scelta del Progetto sta a voi
  - Per ogni progetto bisogna consegnare una relazione scritta (pdf) e un link ad un repository git con il codice
  - **La consegna avviene una settimana prima della data dell'esame**

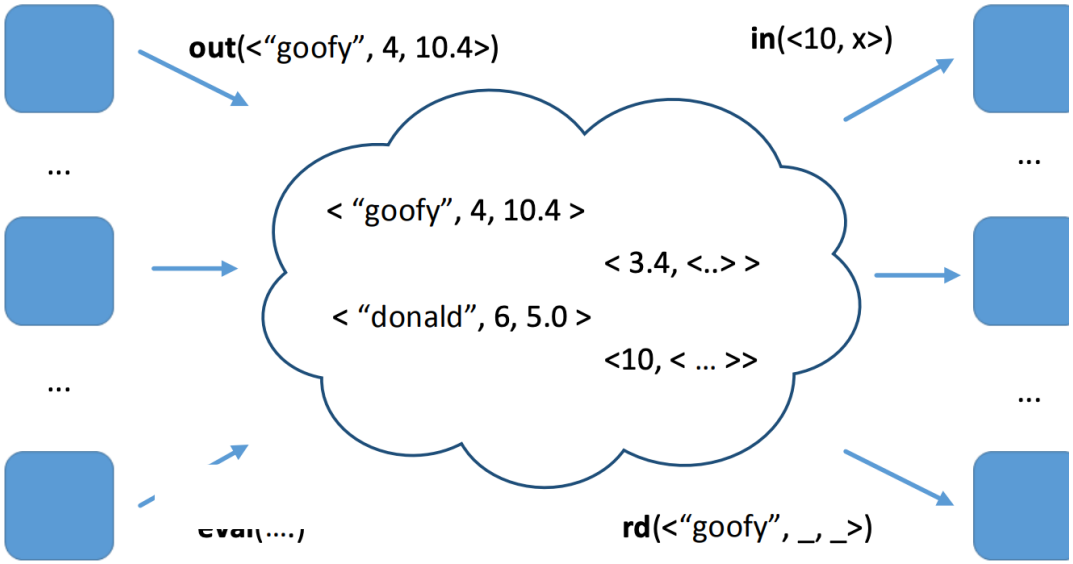
ADCC

Progetto semplice – Tuple Space

Tuple Space

- Un tuple space è un'astrazione di memoria condivisa
- Viene spesso fatta un'analogia con una lavagna (black-board)
  - È uno spazio dove tutti possono leggere e scrivere
- https://en.wikipedia.org/wiki/Tuple_space

Tuple Space



- Out adds a tuple to the tuple space
- in(T) is a blocking operation:
  - it blocks until a tuple matching T shows up
  - In the picture the in can ready any tuple of 2 elements whose first one is 10
  - It consumes/deletes the tuple
- rd(T) is a blocking operation:
  - similar to in(T) but does not consume the tuple

Interface 1/3

- The interface of the TS module must include the following functions:
- `new(name)`
  - Creates a new TS named with name
- `in(TS, Pattern)`
  - Returns a tuple matching the pattern in the TS and deletes if from the TS
  - Blocks if there is no tuple matching
- `rd(TS, Pattern)`
  - Returns a tuple matching the pattern in the TS
  - The tuple remains in the TS
  - Blocks if there is no tuple matching
- `out(TS, Tuple)`
  - Puts the tuple Tuple in the TS

Interface 2/3: timeout

- in(TS, Pattern, Timeout)
  - As in(TS, Pattern) but returns after Timetout
  - {ok,Tuple} or {err, timeout}
- rd(TS, Pattern, Timeout)
  - As rd(TS,Pattern) but return after Timout
  - {ok,Tuple} or {err, timeout}

Interface 3/3

- `addNode(TS, Node)`
  - Adds the Node to the TS, so Node can access to all the tuples of TS
- `removeNode(TS,Node)`
  - Removes a node from the TS
- `nodes(TS)`
  - Tells the nodes on which the TS is visible/replicated

Simplifications

- Just consider flat tuples
- Try to use as much as possible Erlang pattern matching mechanism
- Try to use as much as possible Erlang timeouts

Project report

- You should report on
- Design decisions: why you took such a decision?
  - Motivate every design choice
  - There is no a right or wrong choice
- Implementation details: how you implemented a decision?
  - Add some code snippet
- Measurements:
  - average in/rd/out time
  - Average time to recover when a node fails
- Put Everything on github/bitbucket

ADCC

Progetto difficile – Kademlia

Kademlia

- Kademlia is a distributed hash table for decentralized peer-to-peer computer networks
- It specifies the structure of the network and the exchange of information through node lookups.
- Kademlia nodes communicate among themselves using UDP.
- A virtual or overlay network is formed by the participant nodes. Each node is identified by a number or node ID.
- The node ID serves not only as identification, but the Kademlia algorithm uses the node ID to locate values

Kademlia IDS

- Nodes and entries in the hash have IDS
- IDS are 160bit integers, usually derived by a hash (sha-1)
- Node IDS are given in random way (with a large integer)
- Entries ID are the Hash-value of their name

Kademlia interface 1/2

Should be parametric to k where k is the number of bits used to represent a node / values / buckets

Should be parametric to the time T used to republish the data

Kademlia has four messages/function.

- PING(ID) Used to verify that a node is still alive.
- STORE(Key,Value) Stores a (key, value) pair in one node.
- FIND_NODE(ID) The recipient of the request will return the k nodes in its own buckets that are the closest ones to the requested key.
- FIND_VALUE(Key) Same as FIND_NODE, but if the recipient of the request has the requested key in its store, it will return the corresponding value.
- Each message includes a random value from the initiator. This ensures that when the response is received it corresponds to the request previously sent

Kademlia Interface 2/2

- JOIN
- When joining the network
  - If there is no node / actor then the actor become the bootstrap node
  - otherwise you have to use the PID / ID of another existing node

Protocol

- To ensure data persistence every node periodically publishes its own data (previously published)
- This mechanism is useful to
  - Avoid data loss even in presence of nodes leaving the network
  - Consider the join of new nodes in the network
- Every pair (K,V) is published once per T seconds

Simplifications for Kademlia

- Use PIDs instead of IPs
- Erlang has a strong library for bit-based operations
  - Use xor (exclusive or)
- Each ”node” is an actor
- Key and values:
  - For values use simple types (strings, atoms, numbers)

Kademlia interface 2/2

Join function

- If there is no node / actor then the actor become the bootstrap node
- otherwise you have to use the PID / ID of another existing node

References for Kademlia

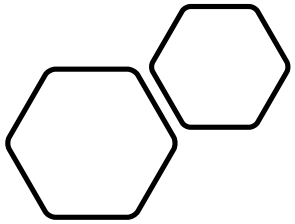
Riferimenti utili

- http://pages.di.unipi.it/ricci/20-03-09-Kademlia.pdf
- https://en.wikipedia.org/wiki/Kademlia
- https://medium.com/coinmonks/a-brief-overview-of-kademlia-and-its-use-in-various-decentralized-platforms-da08a7f72b8f
- https://pub.tik.ee.ethz.ch/students/2006-So/SA-2006-19.pdf

Project report

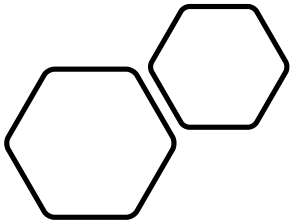
- You should report on
- Design decisions: why you took such a decision?
  - Motivate every design choice
- Implementation details: how you implemented a decision?
  - Add some code snippet
- Measurements:
  - average time of lookup
  - average time of lookup in presence of node failure (up to k-1 nodes per key)
  - average time for joining the network
  - Stress the implementation with several (~1000) nodes in the net

Silver rule



For any questions/doubts use the forum

The golden rule



- There are several implementations on the net/git hub of Tuple Space and Kademlia
- The final goal is not to have a working (copied) implementation
- Is to show that you mastered Erlang enough so to

Have fun

The real golden rule

No bugs