Distributed Systems: Assignment 4
Middleware: Distributed Locks

Petru Eles, Sergiu Rafiliu, and Ivan Ukhov
{firstname.surname}@liu.se

January 14, 2013

1 Introduction

In a distributed system, peers typically share common resources, and they have to perform various operations on those resources simultaneously, *e.g.*, reading/writing from/to a distributed database (our scenario). Consequently, as it is the case with a multi-threaded application, such resources should be protected since several peers may try to modify them at the same time, potentially leaving these resources in corrupted states. The goal of the assignment is to implement an algorithm for distributed mutual exclusion that prevents this undesired behavior. We shall focus on Ricard-Agrawara's second, *i.e.*, token-based, algorithm [1]. Loosely speaking, using this algorithm, peers will be able to uniquely determine—by passing to each other, according to certain rules, a so-called token—which of the peers is allowed to enter its critical section, *i.e.*, to operate on the common resource guarded by the token. This setup is depicted in Figure 1. Apart from the material given in [1], you might also want to refresh your knowledge on logical clocks described in [2] as they are an essential part of Ricard-Agrawara's algorithm.

2 Your Task

2.1 Preparation

Download [3] and extract the source code for the assignment. In the archive, you will find the following files:

- `lab4/mutexPeer.py` — the main application (no changes are needed);
- `lab4/test.sh` — a shell script that you can use for testing;
- `modules/Server/lock/distributedLock.py` — the distributed lock (should be modified);
- `modules/Server/peerList.py` — the same as for Assignment 3 (overwrite with your implementation);
- `modules/Common/nameServiceLocation.py` — the same as for Assignment 2;

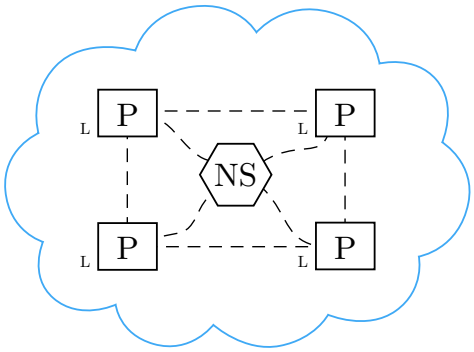

Figure 1: A name service (NS) and several peers (P) enhanced by a distributed locking mechanism (L).

- `modules/Common/objectType.py` — the same as for Assignment 2 (overwrite with your implementation);
- `modules/Common/orb.py` — the same as for Assignment 2 (overwrite with your implementation);
- `modules/Common/wrap.sh` — the same as for Assignment 1.

As usual, you should reuse your implementation from the previous assignment and overwrite the corresponding files listed above. Read and understand the code in `mutexPeer.py` and `distributedLock.py`.

Run several instances of the main application, `mutexPeer.py`, in parallel and try to acquire the lock in all of them simultaneously:

```
$ python mutexPeer.py -t ivan
Choose one of the following commands:
    l  ::  list peers,
    s  ::  display status,
    a  ::  acquire the lock,
    r  ::  release the lock,
    h  ::  print this menu,
    q  ::  exit.
ivan(0) : RELEASED > a
Trying to acquire the lock...
ivan(0) : LOCKED   >
...
ivan(1) : LOCKED   >
...
ivan(2) : LOCKED   >
...
```

As you can see, all the peers have successfully entered their critical sections at the same time; in other words, our distributed lock does not work.

2.2 Implementation

Your task is to implement Ricard-Agrawara's token-based algorithm for distributed mutual exclusion. Read the description of the algorithm given in [1] and complete the following functions of the class `DistributedLock` stored in the module `distributedLock.py`:

- `initialize` — using a populated peer list, the function initializes the state of the lock for the current peer (the token should be initially given to strictly one of the peers).
- `destroy` — called when the current peer leaves the system; if the peer has the token, the token should be passed to someone else.
- `register_peer` — called when some other peer (not the current one) joins the system; the peer should be properly taken into account.
- `unregister_peer` — called when some other peer (not the current one) leaves the system; the peer should be properly removed from consideration.
- `acquire` — called when the current peer tries to acquire the lock.
- `release` — called when the current peer releases the lock.
- `request_token` — called when some other peer requests the token from the current peer (should the current one have the token or not).
- `obtain_token` — called when some other peer gives the token to the current peer.

You implementation is required to maintain the correct behavior of Ricard-Agrawara's algorithm in the case when peers dynamically join and leave the system. For simplicity, you may assume that the peer holding the token never disappears (*e.g.*, due to a crash) without passing the token to someone else.

3 Conclusion

In this assignment, you have implemented one of the algorithms for distributed mutual exclusion. As motivated in the introduction, such an algorithm is a crucial component of a distributed system as it ensures the consistency of shared resources present in the system. In the next, final assignment, you will integrate all your achievements, obtained so far, into a robust distributed database, wherein an arbitrary number of peers will be maintaining the content of the database, and an arbitrary number of users will be able to transparently connect to this distributed database and to safely perform the needed operations.

References

[1] http://www.ida.liu.se/~TDDD25/lecture-notes/lect6-7.frm.pdf.

[2] http://www.ida.liu.se/~TDDD25/lecture-notes/lect5.frm.pdf.

[3] http://www.ida.liu.se/~TDDD25/labs/assignment4.zip.