Seminar Report: Muty

Lluís Marquès, Marc Catrisse, Èric Casanovas

April 4, 2019

Upload your report in PDF format.
Use this LaTeX template to format the report.
A compressed file (.tar.gz) containing all your source code files must be submitted together with this report[1].

1 Introduction

Introduce in a couple of sentences the seminar and the main topic related to distributed systems it covers.
En aquesta pràctica disposem d'un conjunt de treballadors que han de sincronitzar-se i decidir, de manera aleatoria (a la primera part) qui té accés al lock distribuit en cada moment.
Hi distingim 3 estats als locks:

- Open: espera que un treballador demani acces a lock amb un request o faci un take per apropiar-se del lock. Quan rep un take, fa multicast d'aquesta petici o a la resta d'instàncies de lock i entra en estat de 'waiting'.D'altre banda, si rep un 'request', respon amb un 'ok'.
- Waiting: El lock espera missatges 'ok' de la resta de processos en el sistema per a entrar a la zona crítica.
- Held: El lock es troba dins la zona crítica. En aquest cas, segueix revent els request de la resta de locks i els emmagatzema. Quan faci 'release' enviarà un 'ok' a tothom.

En cas de produir-se un 'deadlock', el lock es desbloquejarà automàticament després un temps per mitjà d'un 'release', tot i que no estigui en held.
El lock original és propens a tenir deadlocks i és usat a la primera part de la pràctica. D'altre banda, a les parts 2 i 3 definim noves maneres per a decidir una prioritat dins els 'locks', amb dues maneres diferents d'afrontar el problema:

- lock2.erl: usa IDs als locks per tal de determinar una prioritat entre els treballadors alhora d'accedir a la zona crítica. El lock amb ID més baix, té més prioritat.
- lock3.erl: Utilitza el rellotje de Lambert per afrontar la disputa per la zona crítica, però en cas d'empat, s'usa el ID del lock per decidir la prioritat més alta, un altre cop, ID més petit implica major prioritat.

[1] Describe in the report any design decision required to understand your code (if any)

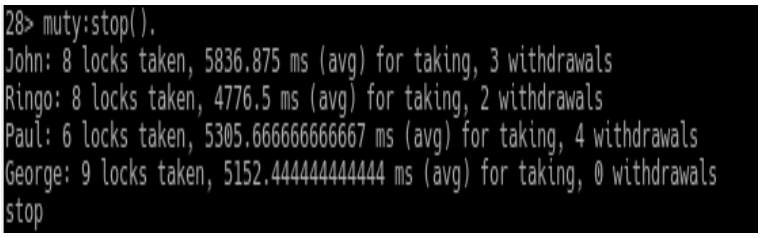
2 Experiments

Provide evidence of the experiments you did (e.g., use screenshots) and discuss the results you got. In addition, you may provide figures or tables with experimental results of the system evaluation. For each seminar, we will provide you with some guidance on which kind of evaluation you should do.

2.1 Part 1

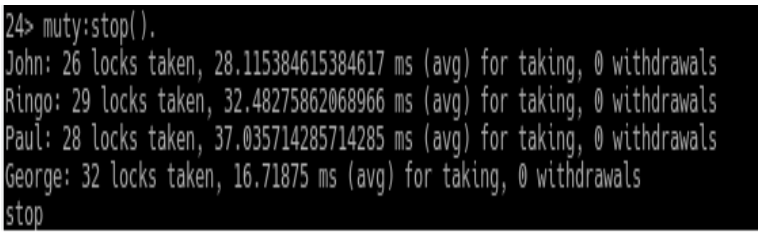
1. Make tests with different Sleep and Work parameters to analyze how this lock implementation responds to different contention.
   Hi hem distingit diferents casos segons els valors que li passem com a paràmetres a l'execució:

   - Quant més alt sigui el valor de Work respecte Sleep, més probable que es produeixin "falsos deadlocks".
     En aquest cas particular hem usat: sleep: 200, work: 5000. Durant la simulació observem que els workers es rendeixen de manera sovint ja que qui té el lock s'hi passa força estona en ell, creant aquesta percepció de 'fals deadlock' a la resta de processos. Hi diem 'fals deadlock' a aquest fet ja que realment no hi ha un 'deadlock' però salta el timeout que allibera el procés, el qual està pensat inicialment, per a tractar i alliberar deadlocks.

     Sleep: 200, Work: 5000, generant falsos deadlocks

   - Si posem un temps de Sleep força major que work, és molt complicat és que es produeixin deadlocks, ja que tots els workers es trobaran en Open, llavors quan volen agafar el lock, quasi sempre estarà lliure. En aquest cas hem fet una simulació amb els següents valors per a obtenir el resultat de la captura següent: sleep: 2000, work: 200.

     Sleep: 2000, Work: 200, els processos sempre troben el lock disponible per entrar-hi

   - Quant menor sigui el numero de Sleep, més probable és que es produeixi un deadlock ja que es més probable que tots demanin els re-

cursos alhora. Hem volgut exemplificar un cas extrem amb: sleep:1 work 200

Sleep: 1, Work: 200, deadlock continunat

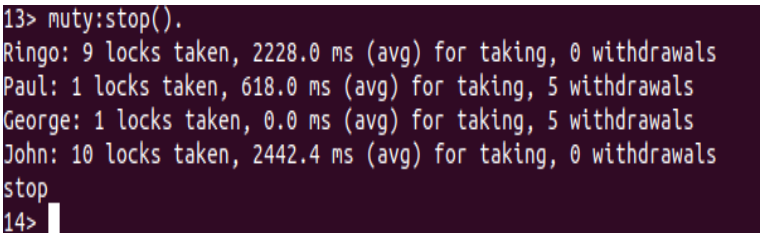
2. Adapt the muty module to create each worker-lock pair in a different Erlang instance (that is, john and l1 should run in a node, ringo and l2 in another, and so on). Remember how processes are created remotely, how names registered in remote nodes are referred, and how Erlang runtime should be started to run distributed programs.

** VEURE CODI ADJUNT: mutyRemote.erl **

2.2 Part 2

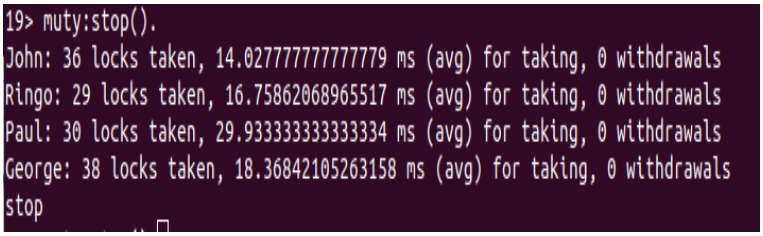
En aquesta segona part millorem el lock per evitar deadlocks i consisteix en que quan 2 workers intentin entrar a la regió crítica entri el que tingui l'id més petit, així evitem que els workers es quedin tots bloquejats.

- En aquest primer cas hem colocat un Sleep petit i un Work gran, i com podem observar a la següent imatge en John i en George han fet molts més locks que els altres 2. Això es degut a que són els que tenen el id més petit i tenen prioritat sobre la resta, a més a més al ser el Sleep petit i el work gran es quedaran normalment menys estona esperant per tornar a demanar el recurs que treballant amb el recurs, fet que explica que només 2 workers siguin els que han agafat el recurs de la regió crítica.



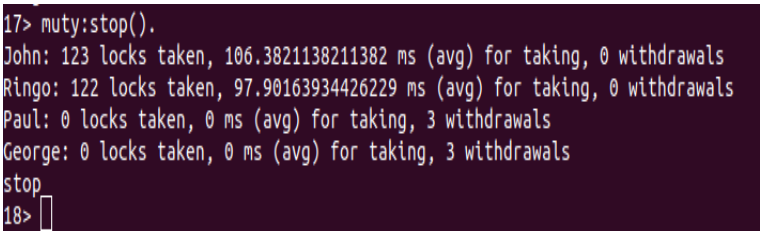
Sleep: 200, Work: 5000

- Aquest segon cas Observem com hi ha una repartició més equitativa dels locks que ha agafat cadascú. Això es degut a que com que el temps de sleep es gran i el de work petit, quan algú agafa el recurs i acaba es queda esperant en sleep molt temps i dona temps a que tots els demes agafin el recurs i facin la feina.



Sleep: 2000, Work: 200

- Al darrer cas, és exactament igual al 1r, ja que al ser el temps de sleep molt petit (en aquest cas excessivament petit) no dona lloc a que qui té prioritat baixa pugui agafar el recurs.

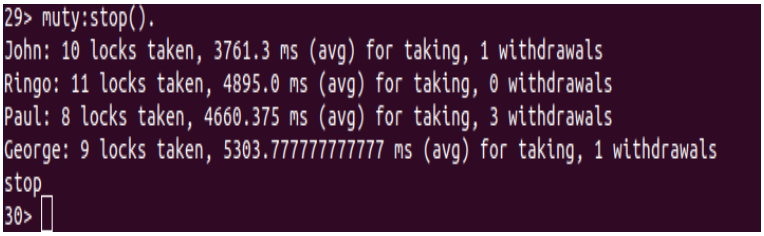


Sleep: 1, Work: 200

2.3 Part 3

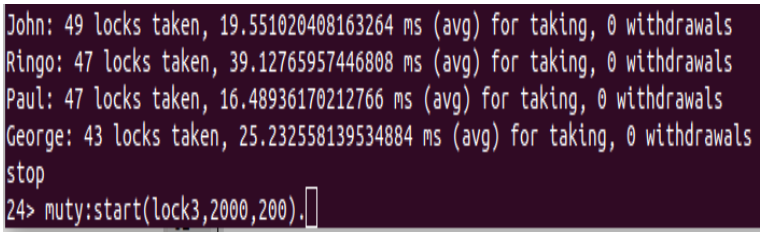
Per millorar el rendiment global del lock2 hem implementat una tercera versió basada en l'ordre de request. Per poder determinar quin procés ha fet el request abans utilitzarem el temps de lamport.
A diferencia del lock2, si en fase de waiting es rep un request, el primer que es farà serà comparar el temps de lamport de la request amb el seu propi. El procés sempre donarà prioritat al temps menor i desempatarà amb la prioritat del procés, com al lock2.
Aixó si, hem de tenir en compte que el temps de lamport només s'incrementarà quan el worker vulgui accedir a la zona d'exclusió mútua.

- En aquest cas observem que el nombre de rendicions es menor al lock2 i lock1, ja que actualment l'accés es reparteix de manera més equitativa. Evitant que un procés sufreixi starvation i es rendeixi.



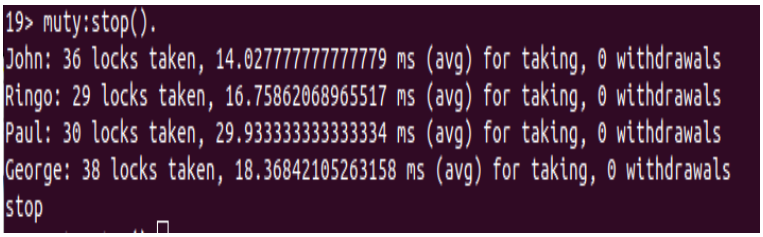
Sleep: 200, Work: 5000

- En aquest cas obtenim un resultat molt semblant al del lock1 i 2, com que el temps de sleep es molt gran hi han poques probabilitats de que tots els procesos demanin accés alhora.



Sleep: 2000, Work: 200, els processos sempre troben el lock disponible per entrar-hi

- Si augmentem el risc de conflicte l'algorisme funciona correctament, es manté la proporcionalitat dels accessos, gràcies al temps de lampert, i no hi han deadlocks.



Sleep: 1, Work: 200

3 Open questions

3.1 Open Questions Part 1

1. What is the behavior of this lock when you increase the risk of a conflict?
   Al afegir un sleep i un work molt petits augmenta molt la probabilitat de deadlock, ja que hi han moltes probabilitats de que més d'un procés es desperti alhora i tracting d'entrar al lock, generant un 'deadlock'.

3.2 Open Questions Part 2

1. Justify how your code guarantees that only one process is in the critical section at any time.
   El nostre codi farà que únicament 1 proces entri degut a que només se li donarà el OK a un dels processos quan estiguin demanant fer un lock del recurs. Però amb una petita excepció que podria donar-se i s'ha de tenir en compte que es quan un procés vol accedir al bloqueig i ja ha reconegut un altre procés amb menor prioritat que encara recopila missatges "OK" i pot rebre un "OK" del node prioritari i que entressin 2 nodes al lock. Aquest fet fa que no respecti el happened-before order. Tampoc es respecte el liveliness mab sleeps bastant mes petits que work.

2. What is the main drawback of lock2 implementation?
   El problema principal d'aquesta implementació com hem vist a l'experiment es que podrien haver alguns nodes que no entressin mai o entressin molt poc a fer el lock. Això provoca un desbalanceig que no ens interessa.

3.3 Open Questions Part 3

1. Note that the workers are not involved in the Lamport's clock. According to this, would it be possible that a worker is given access to a critical section prior to another worker that issued a request to its lock instance logically before (assuming happened-before order)? (Note that workers may send messages to one another independently of the mutual exclusion protocol). No, ja que el valor de clock del lock només s'actualiza quan el worker vol accedir a la regió d'exclusió mútua. Per tant la resta de missatges que pugui rebre el worker no afectarà al clock de lamport d'aquest.

4 Personal opinion

Provide your personal opinion of the seminar, indicating whether it should be included in next year's course or not.

Ens ha semblat un seminari força interessant on hem vist un cas pràctic on podem tractar de solventar el problema de deadlocks que es poden produïr en un sistema distribuit. El principi resultaba una mica confós i pot ser poc intuitiu.