Programació Concurrent i en Temps Real

Exercicis programació distribuïda

Sebastià Vila-Marta

25 de novembre de 2026

1 Exercicis. El banc

1. Implementeu un procés Erlang que correspon a un banc amb tres comptes. Cada compte s'identifica amb un àtom (`compteA`, `compteB`,...). Les operacions bàsiques, que corresponen als missatge que pot rebre el procés, són:

   consulta de la quantitat dipositada en un compte.

   ingrés d'una quantitat en un compte.

   reintegrament d'una quantitat d'un compte. Com a prerequisit, el compte ha de tenir un dipòsit suficient. Si no el té l'operació torna error.

   Un missatge corresponent a una operació, fa que el banc torni un altre missatge amb un codi que indica com ha acabat l'operació: `ok`, `nomoney`. Mentre el procés és viu, conserva la quantitat dipositada. Quan el procés arrenca, els comptes s'inicialitzen amb 0. Els comptes s'implementen sobre un diccionari en el que la clau és el nom del compte (un àtom).

   Com sempre, implementeu també una funció `start()` que crea el procés bancari i retorna el seu `PID`.

2. Afegiu al mateix mòdul un API amb una funció per a cada operació que, envia el missatge apropiat i espera el missatge de tornada. Aquestes funcions us permetran provar fàcilment el funcionament del banc.

3. Opcionalment, afegiu al vostre banc una bateria de tests automàtics. Podeu fer servir, per exemple, «Erlang Common Test» com a framework de testing.

2 Exercicis. Banc distribuït elemental.

1. Recupereu la implementació d'un banc trivial feta a classe de teoria. Proveu-la.

2. Implementeu l'ultrabanc amb tres nodes i proveu-lo.

3. Amplieu els bancs trivials amb un mecanisme de retard que faci que una operació no sigui immediata. Apliqueu l'estratègia de la Weibull(2,2) que vàrem comentar a classe.

4. Esteneu l'exercici anterior per tal que, si el temps de retard es superior a un numero prou gran, posem 5 segons, el sistema simplement no torna resposta.

5. Esteneu el protocol dels bancs trivials afegint el missatge `{self(), avorta}`, que avorta l'operació en curs (si n'hi ha cap). Retorna el missatge {Idb, `true`} si s'ha abortat alguna operació i {Idb, `false`} en cas contrari. Quan una operació sigui llarga, aquest missatge permetrà avortar-la **sense que tingui efecte**. Fixeu-vos bé en tota la casuïstica i procureu que la implementació garanteixi la correctesa en tot cas.

6. Afegiu l'operació d'«ingrés» a l'ultrabanc. Fixeu-vos que heu de garantir que l'operació s'anota a tots els bancs trivials o no s'anota a cap. Això us resultarà complicat, però deixeu volar la imaginació i proveu alguna solució senzilla encara que no sigui perfecta. Quines limitacions té? Quins problemes pot gestionar correctament?

7. Un problema de l'ultrabanc és que podem estar enviar ordres a un banc trivial que, de fet, està mort. Aquestes ordres, doncs sempre causaran conflictes. Una solució podria passar per tenir un procés més dedicat a saber, en tot moment, quins bancs trivials són vius. A tal efecte, el procés enviarà un missatge trivial `{self(), ping}` i esperarà que el banc en un temps breu li contesti `{Idb, pong}`. Si la resposta no arriba en el temps previst, es declararà el banc per mort (definitivament). EL procés mantindrà una llista de bancs vius que el coordinador de l'ultrabanc podrà consultar abans de fer qualsevol operació.

   Aquest esquema presenta alguns problemes. Quins? Alguns d'aquest problemes que detectareu es podran resoldre amb certs mecanismes especials d'Erlang per aquesta finalitat.

8. Modifiqueu l'ultrabanc per que abans d'operar consulti els serveis de banc trivial vius.

3 Exercicis. Banc distribuït amb líder substituïble 1

En el banc distribuït elemental, el fet de tenir un procés coordinador suposa una feblesa important: si el coordinador cau, el banc cau. En aquest exercici s'explora una arquitectura més elaborada, en la que els processos poden fer de coordinador o de banc trivial segons escaigui.

1. Prepareu un directori específic per a aquest exercici.

2. Inicialment assumirem un model(s) sense errors ni caigudes però si amb retards que segueixen una distribució de Weibull.

3. Comenceu dissenyant i preparant un mòdul que permeti introduïr en la resta del codi els retards (i més endavant les pèrdues de missatges i altres fallides) seguint una distribució de Weibull. Penseu detingudament quina és la millor API per a aquest mòdul i implementeu-lo.

4. Dissenyeu un mòdul que implementa un procés «banc» similar al que ja coneixeu. Considereu, però, el següent:

    - Tot banc té un identificador enter únic i coneix els PID de la resta de bancs del sistema.
    - Un banc pot funcionar en dos modes mútuament exclusius: líder i seguidor. En mode líder, el banc pot fer modificacions: reintegraments, ingressos i consultes. En mode seguidor només consultes.
    - Qualsevol banc a l'arrencar ho fa en mode seguidor.
    - Tot banc té una operació per a esdevenir líder. Quan s'activa, informa a la resta de bancs de que esdevindrà el líder i en rebre la confirmació de tots els bancs, es consolida com a líder.
    - En cas que un banc líder rebi una petició de lideratge d'un altre banc, la denegarà i al mateix temps informarà de qui és el líder.
    - Una vegada que un banc ha esdevingut lider, informa a la resta de bancs que és el nou líder. Els bancs en prenen nota.
    - El banc líder, informa de les operacions de modificació dels comptes a la resta de bancs, que les repliquen localment i retornen una confirmació.

5. El sistema en funcionament està format per N bancs similars entre els quals un d'arbitrari fa de líder. Tots poden rebre peticions dels clients. Només el líder pot fer actualitzacions: si un seguidor rep peticions de modificació contesta indicant que no pot i enviant el PID del líder, a fi i efecte que el client s'hi pugui dirigir per a la modificació.

6. Dissenyeu un mòdul que engega un sistema distribuit amb N bancs i retorna els seus PID. El procés d'engegar funciona de la següent manera:

    - Primer es creen N bancs individuals que arrenquen en mode «aïllat».
    - Una vegada creats, l'arrencador els informa de quins bancs constitueixen el sistema enviant-els-hi els PID i els ID. Els bancs canvien de mode «aïllats» a mode «seguidor» i confirmen.
    - En cas que algun no confirmi, la creació del sistema distribuït falla.
    - Una vegada tots els bancs del sistema estan en mode «seguidor», l'arrencador n'escull un com a «líder» i comença el protocol de confirmació de «líder». Si es confirma

com a líder l'arrencador acaba correctament. Altrament, si la confirmació de lideratge falla, l'operació d'arrencar el banc distribuït falla.

7. Escriviu un mòdul per a que els usuaris puguin fer servir amb comoditat el sistema bancari distribuït. Segurament, per fer fàcil l'accés als usuaris, un mòdul amb operacions d'accés fora interessant. També pot jugar-hi un paper els noms simbólics pels processos: investigueu la funció «register()» d'Erlang.

4 Exercicis. Banc distribuït amb líder substituïble 2

Feu una còpia de l'exercici anterior. Endreceu i refactoritzeu el qeu calgui i feu-ho servir com a codi base per aquest exercici.

L'objectiu és millorar el sistema anterior davant un tipus concret de fallida: un crash del procés líder. En aquest cas, el sistema distribuït es quedaria sense part de la funcionalitat.

Els processos seguidors han de seguir l'estat del seu líder i, en cas que aquest falli, algun seguidor s'ha de postular com a nou líder.

Per seguir l'estat del líder, els processos seguidors fan servir el mecanisme de «monitor()» d'Erlang, que està espressament pensat per a aquestes funcions. Investigueu el mecanisme i com s'aplica.

Pel que fa a qui és el següent procés que s'ha d'auto-postular com a líder, sempre s'autopostula el procés amb l'Id més petit: quan aquest descobreix que el lider ha fet crash, automàticament es postula com a líder.

Implementeu aquest nou mecanisme i feu el testing escaient.

5 Exercicis. Rellotges lògics

1. Considereu un sistema distribuït amb tres processos P_1, P_2 i P_3. Els procesos estan involucrats en una colla d'esdeveniments (a, b, c, . . .) que es descriuen a continuació. Noteu que els esdeveniments ocorren en un cert instant de temps WCT (Wall Clock Time) i es mesura en ms.

   - A l'instant O ms WCT

     a: P_2 envia missatge m_1 a $P_1

     b: P_3 llegeix un fitxer d'inicialització

   - A 100 ms WCT

     c: P_1 obre un fitxer que conté un perfil d'usuari

d: P_3 envia un missatge m_2 a a P_2

- A 200 ms WCT

  e: P_1 rep un missatge m_1

  f: P_2 rep un missatge m_2

- A 300 ms WCT

  g: P_1 envia missatge m_3 a P_2 i P_3

- A 400 ms WCT

  h: P_2 rep missatge m_3

- A 500 ms WCT

  i: P_2 envia missatge m_4 a P_1

- A 600 ms WCT

  j: P_1 rep missatge m_4

  k: P_3 rep missatge m_3

Llavors,

(a) Quins dels esdeveniments anteriors tenen una relació «succeí-abans» en el sentit de Lamport? Dibuixeu un graf dirigit amb les relacions causals entre els esdeveniments.

(b) Associeu un timestamp lògic a cada esdeveniment. En cas d'empat, useu els Id dels processos per desempatar. Quin és l'ordre lineal que indueix aquest timestamp?

2. Assumiu que en un sistema quan el rellotge vectorial de P_2 era $(5, 4, 2)$ rep un missatge de P_1. El timestamp del missatge era $(7, 3, 5)$.

   - Que es pot dir del sistema sencer?
   - Què «aprèn» P_2 de la història de la resta de processos?

6 Exercicis: banc distribuit transaccional 1

1. Prepareu un directori específic per a aquesta sèrie d'exercicis que es dedicaran a l'algoritme de 2phC.

2. Implementeu un mòdul `weib` que recull les eines relacionades amb la distribució de Weibull. Afegiu com a mínim dues funcions: una que torna un valor aleatori d'ordre dels mils (per a poder ser aplicat a retards) i una altra `true_sometimes(T)` que torna

`true` amb una probabilitat $T \in [0, 1]$, altrament torna `false`. Feu servir el generador d'aleatoris de distribució uniforme per implementar-la.

3. Implementeu un banc amb les operacions de consultar i ingressar. El banc és transaccional. L'operació d'ingressar desencadena una transacció de l'algoritme 2phC. Això vol dir que l'ingrés no es farà efectiu (només se'n pendrà nota) i s'esperarà que l'algoritme 2phC acabi amb commit per anotar-ho o amb abort per descartar-ho. El banc manté un cert estat. Està normalment en l'estat d'«ocios». Una vegada rebuda una operació d'ingrés, el banc es posa en estat «$\text{wait}_{\text{prepare}}$» indicant que està a l'espera del missatge de «prepare». Quan rep un missatge de «prepare», sempre contesta «true» i es queda en estat «$\text{wait}_{\text{order}}$». En l'estat de «$\text{wait}_{\text{order}}$» pot rebre un missatge de «commit» o «abort». Segons rebi consolida les dades de l'ingrés i confirma amb un missatge d'«ack» per tornar a l'estat d'«ocios».

   De moment no considereu cap tipus d'error ni retard.

4. Implementeu un ubank amb operacions de consultar i ingressar. L'operació d'ingressar ha d'implementar la part del coordinador del protocol 2phC. A tal efecte, quan s'ingressa primer s'envia la operació als col·laboradors i seguidament l'operació de «prepare». S'espera que tots contestin i, segons el que hagin contestat es decideix commit o abort. Es comunica la decissió a tots els col·laboradors i s'esperen els seus acks.

5. Afegiu traces una mica elegants als bancs i a l'ubank, de manera que es pugui seguir com funciona l'algoritme de 2phC.

6. Proveu els mòduls un a un començant pel més simple i acabant per l'ubank. Corregiu i feu que funcionin. Fixeu-vos en si els bancs sempre tenen dades coherents.

7. Afegiu una petita variació. En els bancs, simuleu la possibilitat de que un banc contesti «fals» quan es preguntat pel «prepare». Feu-ho amb la funció `true_sometimes()` que heu implementat. Doneu-li una bona probabilitat d'acceptar la transacció, altrament no aconseguireu tancar gaires transaccions. Per què?

8. Ara, proveu de nou l'ultra banc. Algunes transaccions haurien d'abortar per algun dels bancs no la'ccepta. Podeu provar moltes transaccions seguides amb una construcció tipus:

   ```
   B = ubank:start(ub1).
   L = [ubank:ingressar(B, 10) || _ <- lists:seq(1,40)].
   length([X || X <- L, X == abort).
   ubank:consultar(B).
   ```

   Les traces us haurien d'explicar què ha anat passant. Per exemple:

   ```
   Send ingressa to 3 banc
   Send prepare to 3 banc
   Wait for response to prepare (0,0)
   ```

```
Wait for response to prepare (1,0)
Wait for response to prepare (2,0)
Decide commit
Wait for commit acks (0)
Wait for commit acks (1)
Wait for commit acks (2)
Send ingressa to 3 banc
Send prepare to 3 banc
Wait for response to prepare (0,0)
Wait for response to prepare (1,0)
Wait for response to prepare (1,1)
Decide abort
Wait for abort acks (0)
Wait for abort acks (1)
Wait for abort acks (2)
```

9. Assegureu-vos que els bancs sempre tenen valors coherents fent traces oportunament.

7 Exercicis: : banc distribuit transaccional 2

L'objectiu de l'exercici és simular unes condicions més complicades pels bancs i endurir l'algoritme de 2phC per que també els gestioni correctament.

En aquest cas, suposarem dues complicacions:

1. Els bancs poden trigar a respondre segons una distribució de Weibull(2,2) amb límit de 3 s.

2. Els missatges del banc a l'ultrabanc es poden perdre amb una certa probabilitat.

Per a gestionar aquests escenaris de complexitat caldrà complicar una mica més l'algoritme 2phC tal i com s'explica al capítol de Houmaily i Samaras [1] que us vaig compartir.

1. Per començar, afegiu identificadors de transacció a tots els missatges. Un `Tid` (Transaction id) és un enter que creix monótonament a cada nova transacció. Quan els bancs/ultrabancs reben missatges, només ho fan de la transacció corrent. Les operacions de consulta i ingrés del banc informen al banc de la nova transacció i el banc només les admet si `Tid` és més gran o igual que l'anterior `Tid` que coneixien. El `Tid` permet saber si els missatges rebuts per un banc corresponen a transaccions antigues.
   Afegiu guardes en el `receive` del banc per garantir que els missatges rebuts són de la transacció actual. I si en rebeu de transaccions antigues?

2. Afegiu un retard aleatori de Weibull a cada operació dels bancs, de manera que responguin irregularment en el temps sense deixar mai de respondre. Procureu que el retard

mai superi els 3 segons. Si us és millor, afegiu les funcions convenients al mòdul `weib`.

3. Feu que els missatges que envien els bancs «desapareguin» amb una probabilitat petita (que es pugui configurar). És interessant que el banc escrigui una traça quan un missatge es perd. De cara a implementar-ho segurament us ho facilitarà tenir una funció per enviar missatges a l'ubank que tingui aquest comportament.

Ara teniu uns bancs amb dues patologies, que es manifesten de quan en quan segons les probabilitats amb que les heu configurat. Aquests bancs tenen un temps de resposta irregular, a vegades fins i tot molt gran. A més, quan es comuniquen amb l'ubank, es poden perdre missatges. Ara es tracta de veure com es poden considerar i gestionar aquests problemes durant el 2phC.

1. Quan ubank està esperant les respostes al prepare (cas patològic de comunicacions 2, pp. 3205 del capítol), és possible que algun banc no respongui o bé que ho faci molt tard. Detecteu aquesta possibilitat amb un timeout i, si es produeix, feu que ubank directament decideixi avortar l'operació. Preneu un temps de timeout de 2 s. **Nota:** fixeu-vos que es pot avortar per que encara no s'havia pres una decisió.
   Hauríeu de poder veure traces com aquestes:

```
6> ubank:ingressa(B,10).
ubank:ingressa(B,10).
[3] Send ingressa to 3 banc
[3] Send prepare to 3 banc
[3] Wait for response to prepare (0,0)
[3] Wait for response to prepare (1,0)
[3] Timeout to response to prepare. Abort. (1,0)
[3] Wait for abort acks (0)
[3] Wait for abort acks (1)
[3] Wait for abort acks (2)
abort
```

   - Entre claudàtors, el numero de transacció.
   - Entre parèntesi, el numero de bancs que han contestat Si i No

2. El següent cas patològic de comunicacions és més interessant (cas 4 del capítol pp. 3205). Imagineu que ubank ha enviat les ordres de commit/abort però no rep algun ack (per que va massa tard o per que s'ha perdut). Aleshores, ubank no pot donar per acabada la transacció i l'únic que pot fer és insistir: o sia reenviar l'ordre de nou i esperar els ack. Així mateix, els bancs pot ser que hagin fet l'operació i no hagi arribat l'ack o que ni tant sols hagin fet l'operació. Sigui com sigui, si els bancs reben l'ordre de commit/abort i ja ho havien fet, simplement contesten ack. Si per contra no l'havien fet, la fan i contesten ack. Per implementar aquest cas, haureu de retocar tant el banc com l'ultrabanc. L'identificador de transacció us pot ajudar a saber si l'operació ordenada ja s'havia fet o no.

La detecció del problema, com en el cas anterior, es fa en base a timeouts.

3. Endreceu els missatges de log: feu-los més descriptius, afegiu-ne de nous si ajuden, escriviu-los de manera més elegant... Feu tots els possibles per que els logs siguin de bon llegir i entendre.
   Els logs són els el resultat més interessant d'aquest superexercici, atès que deixen veure com l'algoritme 2phC es comporta en circumstàncies molt diferents i com manté la coherència dels bancs.

4. Afegiu una nova operació al ubank `check(B)` que consulta els comptes de tots els bancs i comprova que tinguin el mateix valor. És una operació circumstancial que només és útil per comprovar que no hi ha discrepàcies en els bancs (i.e.: que l'algoritme de 2phC funciona bé).

Amb aquests canvis, haurieu de poder observar logs com els següents:

```
8> ubank:ingressa(B,10).
ubank:ingressa(B,10).
[3] Send ingressa to 3 banc
[3] Send prepare to 3 banc
[3] Wait for response to prepare (0,0)
[3] Timeout to response to prepare. Abort. (0,0)
[3] Wait for abort acks (0)
[3] Timeout to abort order acks. Retry. (0,0)
[3] Wait for abort acks (0)
[3] Wait for abort acks (1)
[3] Wait for abort acks (2)
[3B(3)] Lost response message (ack)
[3B(1)] Resend ack to done abort
abort
[3B(2)] Resend ack to done abort
[3B(3)] Resend ack to done abort
9>
```

```
17> ubank:ingressa(B,10).
ubank:ingressa(B,10).
[125:2U] Send ingressa to 3 banc
[125:2U] Send prepare to 3 banc
[125:2U] Wait for response to prepare (0,0)
[129:2U] Wait for response to prepare (1,0)
[130:2U] Wait for response to prepare (1,1)
[145:2U] Timeout to response to prepare. Abort. (1,1)
[145:2U] Wait for abort acks (0)
[145:2U] Wait for abort acks (1)
```

```
[150:2U] Wait for abort acks (2)
[165:2U] Timeout to abort order acks. Retry. (2,0)
[165:2U] Wait for abort acks (0)
[165:2B(1)] Resend ack to done abort
[165:2U] Wait for abort acks (1)
[167:2U] Wait for abort acks (2)
[180:2B(2)] Resend ack to done abort
[180:2B(2)] Lost response message (ack)
[182:2U] Timeout to abort order acks. Retry. (2,0)
[182:2U] Wait for abort acks (0)
[182:2B(1)] Resend ack to done abort
[182:2U] Wait for abort acks (1)
[190:2B(2)] Resend ack to done abort
[190:2U] Wait for abort acks (2)
[193:2B(3)] Resend ack to done abort
abort
[208:2B(3)] Resend ack to done abort
```

Noteu com hi ha infinitat de possibilitats i que fer anàlisis a base de «prova-i-error» és totalment inviable.

8 Exercicis: : banc distribuit transaccional 3

L'objectiu en aquests exercicis és afegir algunes condicions patològiques més al sistema anterior.

1. Revisa els missatges de traça i fes-los tant clars, informatius i elegants com sigui possible.

2. Com a aclariment, fixeu-vos que estem assumint que tot el sistema funciona amb transaccions serialitzades (una darrera l'altra en el temps). És a dir, mentre una transacció està en marxa, no se n'executa cap altra. Un model més complicat és el que admet concurrència de transaccions. Per exemple, permet que mentres la transacció N s'està confirmant (aplicant el 2phC), es comenci a executar la transacció $N + 1$. Noti's que una i altra transaccions no han pas de recaure sobre les mateixes dades, la qual cosa demostra que pot tenir sentit executar transaccions en paral·lel.

3. Afegirem un retard de resposta semblant al que ja té el banc a l'ubank. Això és un retard aleatori amb distribució de Weibull(2,2) i limitat a 3 s. Amb aquesta modificació els missatges que envia l'ubank als banks poden patir endarreriments o, fins i tot, perdre's.

4. En el banc preveurem algunes patologies per respondre a la modificació anterior. La primera correspon a la patologia [first, pp. 3205] del capítol i es dona quan el banc

espera el missatge de «prepare» però per alguna raó no arriba. En aquest cas, el banc decideix unilateralment avortar la transacció.

5. En realitat, aquesta transacció que hem decidit avortar unilateralment en el punt anterior és possible que no sigui considerada avortada per l'ubank. Per exemple, és possible que senzillament el misatge de «prepare» s'hagi enviat però s'hagi endarrerit i acabi arrivant
   Conseqüentment, qualsevol «prepare» posterior sobre aquesta transacció l'ha de respondre amb «abort». Igualment, qualsevol ordre d'«abort» sobre aquesta transacció l'ha de respondre amb «ack» immediatament.
   Es pot assumir que si arriba una nova ordre de l'ubank (i per tant amb nou identificador de transacció) ja no arribaran missatges de transaccions anteriors.

6. La següent patologia que cal considerar correspon a la [third, pp. 3275] del capítol. Es tracta de la possibilitat que el missatge de «commit» o «abort» de l'ubank no arribi i es dispari el timeout del banc. En aquest cas, el banc no pot fer altra cosa que esperar-se a que arribi: no pot prendre cap decisió autònoma amb garantia que sigui coherent. L'estratègia consistirà a preguntar a l'ubank sobre la decissió fins que aquest contesti.
   Noteu que això genera un nou tipus de missatge del bank a l'ubank que cal gestionar. L'ubank només ha d'escoltar aquest missatge si correspon a la transacció actual, ha enviat les ordres de «commit»/«abort» i està pendent de rebre els «ack».

Referències

[1] Yousef J. Al-Houmaily i George Samaras. "Two-Phase Commit". A: *Encyclopedia of Database Systems*. Ed. de Ling Liu i M. Tamer Özsu. 2a ed. Springer, 2018, pàg. 3204 - 3209. DOI: https://doi.org/10.1007/978-1-4614-8265-9.