Egy konszenzus protokoll (Paxos) modellezése és verifikációja

Készítette: **Izsó Benedek**

2011.

Tartalom

1 A konszenzus protokoll leírása 1
2 A protokoll modellezése 2
3 A követelmények ellenőrzése 2
4 Finomságok 3
5 Hivatkozások 3

1 A konszenzus protokoll leírása

Elosztott rendszerekben fontos kritérium, hogy bizonyos helyzetekben a csomópontok egy közös értékben egyetértsenek. Ezen érték megbeszélésére használhatóak a konszenzus protokollok. A feladat egy közismert [1,2], Google-nél is használt [3] konszenzus protokoll egyszerűsített változatának modellezése és verifikálása.

Az algoritmusban háromfajta szereplő van: *Javasló (Proposer)*, *Elfogadó (Acceptor)* és *Tanuló (Learner)*, melyek saját azonosítóval rendelkeznek.

1 A Javasló legfeljebb 2 óránként (de legalább 3 másodperc után) új *értéket* javasol, amit a 0, 1, 2 vagy 3 számok közül véletlenszerűen sorsol ki.

1.1 Ehhez először minden Elfogadónak elküld egy *sorszámot* (prepare), és megígérteti velük, hogy annál kisebb sorszámú javaslatot ezután már ne fogadjanak el. Erre az Elfogadó nem biztos, hogy válaszol, de ha válaszol (promise), és már korábban fogadott el értéket, akkor azt és annak sorszámát is közli. Ha a Javasló nem tudja elküldeni az üzenetet, vagy az Elfogadó nem válaszol, akkor 5 másodperces időtúllépés után a Javasló a következő Elfogadóval veszi fel a kapcsolatot.

1.2 Ezután, ha az Elfogadók többsége ígéretet tett a Javasló által küldött sorszámra, akkor a Javasló új *értéket* fogadtat el. Ha az Elfogadók közül még senki nem fogadott el értéket, akkor a Javasló saját sorsolt értékét fogadtatja el; egyébként köteles a válaszban megkapott legnagyobb sorszámú értéket elfogadtatni. Ez úgy történik, hogy a Javasló az összes Elfogadóval üzenetben közli, hogy fogadja el (accept) értékét. Amennyiben nem sikerül elküldeni az értéket (és a sorszámot), 5 másodperc időtúllépés után a következővel próbálkozik. Az elfogadást csak közli, erre választ már nem vár. A teljes folyamat után a Javasló alapállapotba tér vissza.

2 Az Elfogadó tehát veszi a Javaslók üzeneteit, de ha már egy nagyobb sorszámra ígéretet tettek, akkor a *kisebb sorszámú* üzenetet annak megfelelően mindig *figyelmen kívül* hagyják.

2.1 A Javasló ígéret előíró kérésére (prepare) válaszolnak (promise). Amennyiben van már elfogadott értékük és ahhoz tartozó sorszámuk, akkor azt is megadják. Ám hiba esetén előfordulhat, hogy az Elfogadó a kérésre mégsem küld választ, még ha a sorszám megfelelő, akkor sem.

2.2 Amikor egy Javasló sorszámának és értékének elfogadására szólít fel egy Elfogadót (accept), akkor az Elfogadó (ígéretével konzisztens sorszám esetén) elfogadja, majd legfeljebb 1 másodperces időközökkel elválasztott (learn) üzenetet küld a Tanulóknak, hogy jelentse nekik, hogy ő már elfogadott egy értéket.

3 A Tanuló hiba nélkül veszi az Elfogadók üzeneteit (learn), és amennyiben egy érték elfogadásáról több üzenetet kapott, mint az Elfogadók fele, akkor azt megtanulja.

2 A protokoll modellezése

A protokoll modellje a csatolt *paxos.xml* fájlban található.

3 A követelmények ellenőrzése

Az alábbi követelmények ellenőrzése egy olyan konfigurációban történt, ahol a rendszerben két Javasló, két Elfogadó és két Tanuló van.

1. Lehet olyan eset, amikor a Javasló nem saját értékét fogadtatja el, hanem egy valamely Elfogadó által korábban elfogadott értéket.

   E<> exists (i : int[0, NP-1]) Proposer(i).DoAccept && Proposer(i).maxn != -1

2. Lehet olyan eset, amikor valamely Tanuló nem tanult meg semmit, de már mindegyik Javasló próbálkozott egy javaslattételi körrel.

   E<> exists (i : int[0,NL-1]) Learner(i).learntValue == -1 && forall (i : int[0,NP-1]) Proposer(i).n > 1

3. Ha egy Tanuló megtanul egy értéket, akkor az csakis egy javasolt érték lehet.

   A[] forall (i : int[0,NL-1]) (Learner(i).learntValue == -1 || exists (j : int[0,NP-1]) Learner(i).learntValue == Proposer(j).newToAccept)

4. Nincs olyan eset, hogy két különböző Tanuló különböző értéket tanul meg.

   E<> forall (i : int[0,NL-2]) Learner(i).Stable && Learner(i+1).Stable && Learner(i).learntValue != Learner(i+1).learntValue && forall (j : int[0,NA-1]) Acceptor(j).Stable

   (Az így formalizált kritérium nem teljesül.)

4 Finomságok

- A „dueling proposers” problémája nincs megoldva. Azaz ha egyik Javasló elfogadtatja az ő sorszámát, aztán feléled a másik és az is elfogadtatja a nagyobb sorszámát, majd megint az egyik a még nagyobb sorszámát, stb.; de semelyik sem jut el az elfogadtatás fázisba (vagy legalábbis nem veszik figyelembe az Elfogadók), akkor a Tanulók nem fognak semmit sem megtanulni (nem lesz közös érték).
- Lehetséges, hogy egy körig még nem alakul ki konszenzus, de ez akkor van, ha az Elfogadók mind hibásak és már a javaslatot sem veszik.
- A Tanuló modellje csak korlátozottan helyes, mert ezzel az implementációval nem tudja megállapítani, hogy az Elfogadók többsége tényleg elfogadott-e egy értéket. Ez a megadott időzítéssel és a megadott konfigurációban (2-2-2 példánnyal) nem fordul elő. Több példány esetén az ellenpélda: Az első Javasló megígérteti az Elfogadók többségével a sorszámát, de csak eggyel fogadtatja el. Majd jön egy másik Javasló, aki még nagyobb sorszámra kap ígéretet, majd megint az első ami még az annál is nagyobb sorszámra is megkapja az ígéretet. Ekkor az megint ugyanazzal az 1 Elfogadóval fogadtatja el a kialakulandó közös értéket. Ez, ha sokszor lejátszódik, akkor az egyik Elfogadó sokszor tanul meg egy értéket, amit minden egyes alkalommal közöl a Tanulókkal, amik így egy idő után megtanulják.

5 Hivatkozások

1. Leslie Lamport: The Part-Time Parliament. ACM Transactions on Computer Systems 16, 2 (May 1998), pp 133-169, 1998.
   http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#lamport-paxos
   http://research.microsoft.com/en-us/um/people/lamport/pubs/lamport-paxos.pdf
2. Leslie Lamport: Paxos Made Simple. ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001) pp 51-58, 2001.
   http://research.microsoft.com/en-us/um/people/lamport/pubs/pubs.html#paxos-simple
   http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf
3. Mike Burrows: The Chubby lock service for loosely-coupled distributed systems. Proceedings of OSDI '06, the 7th USENIX Symposium on Operating Systems Design and Implementation, Vol. 7. 2006.
   http://labs.google.com/papers/chubby-osdi06.pdf