**Appunti studio Paxos vs Dwork**

**Paxos semplice**

**Introduzione generale**

Si tratta di un algoritmo semplice per la gestione di un sistema distribuito in grado di tollerare errori/problemi. Si tratta di un algoritmo a consenso.

**L’algoritmo a consenso**

Vari processi propongono valori, solo uno viene scelto ed accettato da tutti.

Ci sono 3 ruoli, che possono essere assunti da “agenti”:

1. Proposers: propongono uno (o più) valore/i.
2. Acceptors: accettano uno (o più) valore/i.
3. Learners

Gli agenti possono comunicare tra loro, per decidere (a maggioranza) quale valore accettare.

Requisito 1: un acceptor deve accettare la prima proposta che gli arriva.

Ogni è proposta di valore viene quindi contrassegnata da un numero (univoco) ed un valore.

Requisito 2: se una proposta con un valore viene scelta, allora tutte le proposte accettate con un identificativo maggiore devono avere valore .

Requisito 2a: se una proposta con valore viene scelta, allora tutte le proposte con un identificativo maggiore, accettate da qualsiasi acceptor, devono avere valore .

Requisito 2b: se una proposta con valore viene scelta, allora tutte le proposte con un identificativo maggiore, proposte da qualsiasi proposer, devono avere valore .

Requisito 2c: per ogni ed , se una proposta con valore ed identificativo è presentata, allora c’è un set composto da una maggioranza di acceptors, tale che vale almeno una tra:

1. Nessun acceptor in ha accettato una proposta con identificativo minore di .
2. è il valore della proposta con identificativo più grande tra tutte le proposte con identificativo minore di , prese tra quelle accettate dagli acceptors in .

**Algoritmo base di gestione delle proposte**

Per garantire il rispetto dei requisiti/invarianti stabiliti, l’algoritmo di gestione delle proposte ragiona in questo modo:

1. Un proponente sceglie un nuovo identificativo per la proposta, ed invia una richiesta a tutti gli acceptors in un set scelto, richiedendo in risposta:
   1. Una promessa che non vengano accettate nuove proposte con identificativo minore di .
   2. La proposta accettata con l’identificativo più alto, inferiore ad , se ce ne sono.

Tale richiesta viene chiamata *prepare* request con identificativo .

1. Se il proponente riceve le risposte richiesta da una maggioranza di acceptors, allora può presentare una proposta con un identificativo e valore , dove è il valore della proposta con identificativo maggiore tra le risposte, oppure un valore scelto arbitrariamente dal proponente se nessuna risposta è stata ricevuta.

Il proponente presenta quindi la proposta, inviandola ad un set di acceptor per l’accettazione. Viene chiamata *accept* request.

**Algoritmo finale di gestione delle proposte**

Requisito P1a: un acceptor può accettare una proposta numerata se non ha risposto ad una *prepare* request con un identificativo maggiore di .

Dopo opportune considerazioni sulla gestione delle accettazioni da parte degli acceptors, si ottiene l’algoritmo seguente, composto da 2 fasi:

Fase 1

1. Un proposer seleziona un identificativo ed invia una *prepare* request con numero alla maggioranza degli acceptors disponibili.
2. Se un acceptor riceve una *prepare* request con un numero maggiore di qualsiasi altra *prepare* request a cui abbia già risposto, allora risponde a questa con una promessa, di non accettare alcun’altra proposta numerata meno di , corredata dalla proposta con identificativo maggiore tra quelle accettate (se esiste).

Fase 2

1. Se il proposer riceve una risposta alle sue *prepare* requests (con identificativo ) da una maggioranza di acceptors, allora invia un’*accept* request ad ognuno di quegli acceptors. Tali richieste sono per una proposta con identificativo e valore , dove è il valore della proposta con identificativo maggiore tra le risposte, o un valore arbitrario se nessuna risposta.
2. Se un acceptor riceve un’*accept* request per una proposta numerata , quest’ultimo accetta la proposta, a meno che non abbia già risposto ad una *prepare* request con un identificativo maggiore di .

Essendo prevista la possibilità di ignorare richieste, in caso di situazioni particolari descritte, magari è opportuno informare il mittente di tale scelta.

**Imparare il Valore scelto (Learners)**

Per imparare un Valore scelto, i learner devono venire a sapere che la proposta è stata accettata dalla maggioranza degli acceptors.

Vengono valutare più opzioni per gestire la cosa:

* Tutti gli acceptors avvisano tutti i learner che un valore è stato accettato, ma non è molto efficiente in termini di performance.
* Tutti gli acceptors avvisano un unico learner che un valore è stato accettato e questi poi avvisa il resto dei learner. Questo migliora un po’ le performance, ma il single point of failure è ovviamente un problema.
* L’opzione precedente con un set scelto di learners potrebbe essere la soluzione migliore, e ci si trova quindi in una situazione di bilanciamento tra affidabilità e performance basato sulla cardinalità del set.

Anche con queste valutazioni eventuali fallimenti lato acceptor potrebbero creare problemi e perdita del valore accettato.

**Progressi nello studio**

Per garantire un funzionamento adeguato del sistema, considerando che è quasi sempre possibile porlo in una situazione tale per cui i proposer si mettono in difficoltà tra di loro, è stata pensata una soluzione a singolo proposer eletto.

**Implementazione dell’algoritmo**

Un’implementazione adeguata dovrebbe considerare le seguenti caratteristiche:

* L’algoritmo prevede un network di processi che svolgano tutti e 3 i ruoli precedentemente descritti.
* L’algoritmo sceglie un leader, che svolge il ruolo di proposer eletto e learner eletto.
* Il funzionamento dell’algoritmo è quello descritto in precedenza.
* È necessario uno spazio di archiviazione stabile, per mantenere le informazioni utili agli acceptor anche in caso di fallimento. Un acceptor memorizza la propria risposta nello spazio di archiviazione prima di inviarla.
* È necessario descrivere un meccanismo che garantisca che non vi siano 2 proposte presentate con lo stesso identificativo.

Per l’ultima caratteristica si può considerare che:

* Proposte differenti scelgano il proprio identificativo da set di numeri distinti, così che non possano mai avere lo stesso identificativo.
* Ogni proposta memorizza (nello spazio di archiviazione) il valore della proposta con identificativo più grande che sia stata presentata, ed inizializzano la fase 1 con un identificativo maggiore di quest’ultimo.

**Implementazione di una macchina a stati**

*Vedi nel paper un esempio interessante di implementazione dell’algoritmo in un contesto concreto e reale*.

**Note aggiuntive derivanti da ulteriori fonti**

* L’algoritmo deriva da un esempio storico del Paxos Parliament.
* Vi è poi una simile applicazione, precedente e meno evoluta nel Sinodo cerimoniale dei sacerdoti. Era basato su ballottaggi e gestione di proposte con implicazioni simili a quelle già discusse.
* Dalla discussione sulla gestione del concilio dei sacerdoti, risultano evidenti molte delle osservazioni nate e discusse nel caso tecnico (molto più chiare nel caso tecnico).
* L’algoritmo di Paxon è adatto a sistemi con modesti requisiti di affidabilità, che non giustificano la spesa di un’implementazione real-time estremamente fault-tolerant.
* Con l’algoritmo di Paxon, il tempo non può essere parte dello stato-macchina. Quindi se avviene un fallimento, può volerci un tempo arbitrariamente lungo per eseguire un comando, e il comando di qualcuno potrebbe venire eseguito prima di un comando che era stato inviato precedentemente.
* Il tempo può comunque utilizzare il real-time per la gestione di incarichi esterni a tempo.
* Il protocollo di Paxon è simile ad un protocollo standard con commit a tre fasi. Entrambi, infatti, utilizzano uno scambio di 5 messaggi tra coordinatori e gli altri membri del quorum:
  + Il commit protocol sceglie uno di due valori (commit-abort) mentre il protocollo di Paxon sceglie un decreto in maniera arbitraria.
  + Per convertire un commit protocol in Paxon, qualcuno deve inviare il decreto nel round iniziale di messaggi.
  + Una decisione di *commit* indica che questo decreto è passato, mentre un *abort* indica che un altro è passato.
* La differenza vera e propria dal commit protocol è che il decreto non viene inviato fino alla fase 2. Questo consente al protocollo parlamentare corrispondente di eseguire la prima fase solo una volta per tutti i decreti, quindi è sufficiente lo scambio di soli 3 messaggi per decreto.
* Altri protocolli (come quello di Dwork) sono simili, ma utilizzano ballottaggi sequenziali in round separati.