PBFT

Practical Byzantine Fault Tolerance

Leonardo Perugini

leonardo.perugini2@studio.unibo.it

Laurea Magistrale in Ingegneria e Scienze Informatiche

Corso di Sistemi Distribuiti

Marzo 2023

I molti sistemi distribuiti in rete

- forniscono servizi
- devono operare in maniera corretta
- devono operare senza mai interrompersi
  - ! restano vulnerabili ad attacchi, difetti e guasti

- molte tecniche cercano di fare fronte alle cause che potrebbero portare ad interruzioni del servizio o ad un suo funzionamento arbitrario

Consenso nei Sistemi distribuiti

Assenza di centralizzazione

- giungere a decisioni comuni accordandosi di volta in volta
- lo stato del sistema deve rimanere corretto
- in assenza di compromissioni, ogni nodo deve...
  - prendere una decisione in un tempo limitato
  - terminare ogni computazione avviata
  - fornire risultati deterministici

Nodi corrotti

Guasti e compromissioni

! non sono critiche eventualità, ma fasi preventivate dell'esecuzione

Byzantine Faults

- il nodo può smettere di funzionare, omettere alcuni step o fornire risultati arbitrari
  - *Denial of Service*
  - *Operator mistakes*
  - *Software errors*
  - *Malicious attacks*

Practical Byzantine Fault Tolerance

PBFT

- algoritmo di State Machine Replication
- con n nodi nel sistema, un massimo di fallimenti pari a

$$f = \left\lfloor \frac{(n-1)}{3} \right\rfloor$$

3 fondamentali

- *Strong Cryptography*: tecniche crittografiche non revertibili
- *Weak Synchrony*: limitati tempi di ricezione dei messaggi
- *Bounds on failures*: limite massimo al numero di repliche fallate

Crittografia

2 applicazioni

- *Correttezza del contenuto*: messaggi processati mediante funzioni di hash *D* generando codici *digest* che accorpati al contenuto
- *Autenticità del mittente*: messaggi processati mediante algoritmi di cifratura simmetrica generando codici ***MAC*** accorpati al messaggio

- il destinatario verifica correttezza e autenticità processando il messaggio ricevuto e confrontando digest e MAC generati con quelli ricevuti

Comunicazione

Canali comunicativi

- **point-to-point**: ogni messaggio inviato da un nodo *i* ad un nodo *j* contiene un singolo MAC
- **multicast**: ogni messaggio inviato da un nodo *i* a tutti gli altri contiene un *authenticator*

  authenticator array di MAC, uno per ogni replica *j*

Quorum System I

Quorum

- qualsiasi sottoinsieme di repliche che prende parte ad una certa decisione

intersection ogni coppia di quorum condivide almeno una replica corretta in comune

availability è sempre possibile ottenere un quorum in cui non ci sono repliche fallate

Quorum System II

Memoria condivisa

- le informazioni ricevute dagli altri nodi compongono dei *certificati di quorum*, che garantiscono l'avvenuta memorizzazione delle informazioni

strong set di $2f + 1$ messaggi da differenti mittenti, garantisce l'avvenuta memorizzazione delle informazioni nelle repliche del quorum

weak set di $f + 1$ messaggi da differenti mittenti, garantisce l'avvenuta memorizzazione delle informazioni in almeno una replica corretta

Primary-Backup I

Viste

- successione di configurazioni in cui, a turno, una replica svolge il ruolo di primary, mentre le altre agiscono da backup

primary svolge le normali operazioni e assegna alle richieste un progressivo detto *sequence number* e lo comunica ai backup

! rischioso se corrotto

backup svolgono le normali operazioni e monitorano le informazioni comunicate dal primary per rilevare inattività o comportamenti arbitrari

Primary-Backup II

Scelta del primary

- in un sistema formato da n repliche, ogni nodo ed ogni vista vengono rappresentati da un numero intero nel range $\{0, ..., |n| - 1\}$
- il primary di una vista è la replica $p = v \mod |R|$, dove v è l'intero che rappresenta l'attuale vista

ViewChange

- processo che consente alle repliche di concordare il passaggio ad una vista successiva eleggendo un nuovo nodo come primary

Client I

Richiesta

- un client c richiede l'esecuzione di un'operazione effettuando un invio multicast al tutte repliche di una

$$\langle REQUEST, o, t, c \rangle_{\alpha_c}$$

  - o operazione richiesta
  - t timestamp
  - c identificativo del client

! è compito del client calcolare e accorpare alla richiesta il relativo MAC

Client II

Risposta

- il client riceverà dalle single repliche dei messaggi del tipo

$$\langle REPLY, v, t, c, i, r \rangle_{\mu_{ic}}$$

  - v vista corrente
  - t timestamp
  - c identificativo del client
  - i identificativo della replica
  - r risultato

! prima di accettare il risultato r, il client deve ricevere $f + 1$ risposte da repliche differenti, con MAC valido e gli stessi valori di t ed r

Client III

Timer

- avviato al momento dell'invio della richiesta
- limita il tempo d'attesa del certificato

! se il timer scade prima dell'ottenimento del certificato la richiesta viene ritrasmessa

Server I

Stato del servizio

- ogni replica ne mantiene una copia
- composto da
  - valore dello stato del servizio
  - log contenente tutti i messaggi accettati e inviati
  - identificativo della vista corrente

Server II

Protocollo in 3 fasi

- *pre-prepare*
- *prepare*
- *commit*

- pre-prepare e prepare garantiscono il corretto ordinamento delle richieste
- prepare e commit garantiscono l'ordinamento delle operazioni di commit, anche a fronte di cambiamenti di vista

Gestione messaggi di REQUEST

Backup

- autentica e memorizza la richiesta nei propri log

Primary

- una volta autenticata la richiesta le assegna un *seqnum*, dopodichè invia alle altre repliche una

$$\langle PREPREPARE, v, n, D(m) \rangle_{\alpha_p}$$

v vista corrente

n sequence number

$D(m)$ digest

Gestione messaggi di PRE-PREPARE

- una replica che accetta una PRE-PREPARE e possiede la relativa richiesta, effettua un'invio multicast di una

$$\langle PREPARE, v, n, D(m), i \rangle_{\alpha_p}$$

  - v vista corrente
  - n sequence number
  - $D(m)$ digest
  - i identificativo della replica

- ! entrambi i messaggi di PRE-PREPARE e PREPARE vengono loggati

Gestione messaggi di PREPARE

- una replica i accetta messaggi finchè non possiede nei suoi log la PRE-PREPARE e almeno $2f$ PREPARE con uguali n, v e m
  - al verificarsi di questa condizione, ottiene un *prepared certificate*
    - ! attesta che un quorum di repliche è d'accordo nell'assegnare il sequence number n al messaggio m nella vista v
- la replica comunica l'ottenimento del certificato generando un

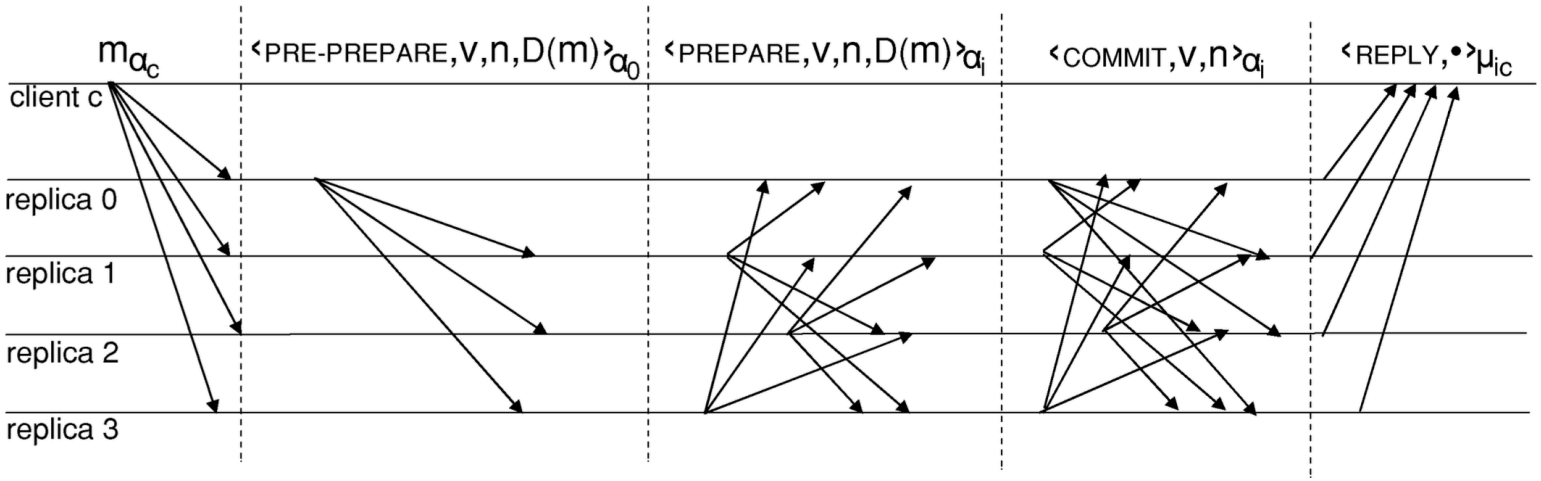
$$\langle COMMIT, v, n, i \rangle_{\alpha_i}$$

che aggiunge ai propri log ed invia in modalità multicast

Gestione messaggi di COMMIT

- una replica *i* accetta messaggi finchè non possiede nei suoi log almeno $2f + 1$ messaggi di COMMIT per il sequence number *n* e vista *v*
  - al verificarsi di questa condizione, ottiene un *committed certificate*
    - ! la replica può ora eseguire la richiesta, posto che abbia terminato l'esecuzione di tutte le precedenti
- al termine dell'esecuzione, la replica invia una Reply al client

Server



Rappresentazione di un normale scenario di esecuzione con 1 client e 4 repliche

Checkpoint system I

- l'avanzamento dell'esecuzione, soprattutto in un sistema con molte repliche, causa l'aumento delle dimensioni dei log
  - ! alcuni log potrebbero essere scartate se non più utili
  - !! alcuni certificati potrebbero essere utili all'avanzamento

Soluzione

- le repliche producono una *prova* che garantisca la correttezza dello stato in un determinato punto dell'esecuzione
  - ? ogni volta che viene eseguita una richiesta con *seqnum* divisibile per un intero K
  - ! i log antecedenti al checkpoint possono essere rimossi

Checkpoint system II

- una replica i che produce o recupera un checkpoint invia alle altre un

$$\langle CHECKPOINT, n, d, i \rangle_{\alpha_i}$$

  - n sequence number dell'ultima richiesta la cui esecuzioneè *riflessa* nello stato
  - d digest
  - i identificativo della replica

- una replica j che raccoglie almeno $2f + 1$ CHECKPOINT con stessi n e d, ottiene uno *stable certificate*.
  - ! tutti i log per richieste antecedenti ad n possono essere rimossi

ViewChange I

Avviare una ViewChange

- un backup i che sospetta della correttezza di p avvia il processo modificando la propria vista in $v + 1$ ed inviando una

$$\langle VIEWCHANGE, v + 1, h, C, P, Q, i \rangle_{\alpha_i}$$

- h sequence number dell'ultimo checkpoint stabile
- C set di coppie $\langle seqnum, digest \rangle$ di ogni checkpoint noto
- P set di informazioni sulle richieste prepared nella replica i
- Q set di informazioni sulle richieste pre-prepared nella replica i
- i identificativo della replica

ViewChange II

ViewChange-Ack

- le repliche accettano le VIEW-CHANGE solo se contengono in P e Q informazioni su viste minori o uguali alla propria.
- le repliche inviano un acknowledgment al primary della nuova vista

$$\langle VIEWCHANGEACK, v+1, i, j, d \rangle_{\mu_{ip}}$$

i identificativo della replica

j mittente del messaggio di VC

d digest del messaggio di VC a cui fa riferimento

- così il nuovo p stabilisce l'autenticità dei messaggi di VC

ViewChange III

Messaggi di New-View

- quando p riceve almeno $2f + 1$ acknowledgment per una VC
  - ottiene un *view-change certificate* per quella VC
  - aggiunge la VC in un set S
  - esegue la *Decision Procedure*
    - ? se la procedura viene portata a termine, p aggiorna il proprio stato e invia una

$$\langle NEWVIEW, v + 1, V, X \rangle_{\alpha_p}$$

- una replica che riceve una NEW-VIEW e possiede la VC, esegue la *Decision Procedure* per determinare i propri risultati
  - se questi coincidono con quelli ricevuti, aggiorna lo stato
  - altrimenti avvia il procedimento per una ulteriore ViewChange

ViewChange IV

Decision Procedure

- procedura necessaria alla determinazione di un set di richieste
  - partendo dal più recente checkpoint conosciuto
  - ogni richiesta R scelta deve rispettare le condizioni A o B:
    - **A** per essere soddisfatta devono essere verificate
      - A1. R era prepared in almeno una replica parte di un quorum nella vista v
      - A2. R era pre-prepared in almeno una replica corretta in una precedente vista e può diventare prepared in $v + 1$
    - **B** esiste un quorum di repliche il quale attesta che nessuna richiesta soddisfa interamente **A**
    - ! viene selezionata una richiesta *no-op* che non modifica lo stato

Proactive Recovery I

- in sistemi con esecuzioni prolungate aumenta la probabilità di superare le f repliche fallate.
- mediante un meccanismo di *recovery* è possibile ripristinare il corretto comportamento delle repliche.
  - ! il numero di repliche fallate totale potrà superare f
  - !! un meccanismo *proattivo* ripristina anche le repliche corrette o fallate con comportamenti insospettabili.

Proactive Recovery II

Ulteriori fondamentali

- *cryptographic co-processor*
  - cifratura e decifratura senza esposizione delle chiavi
  - contatore interno accorpato ad ogni messaggio inviato
- *read-only memory*: chiavi pubbliche mantenute in zone inalterabili
- *watchdog timer*: innesca il meccanismo di Recovery ad intervalli regolari

Proactive Recovery III

Recovery nella replica *i*

1. **reboot**
   - avvio di una VC per $v + 1$
   - salvataggio dello stato e riavvio del sistema
   - caricamento del codice e dello stato
   - aggiornamento delle chiavi con i client
2. **estimation Protocol**: definisce il più alto *seqnum* che *i* potrebbe aver inviato nel caso fosse stata corrotta
3. **recovery requests**: definisce il *recovery point* di *i* e aggiorna le chiavi con le altre repliche
4. **state check ad recovery**: recupero e controllo di correttezza dello stato mancante o non aggiornato dalle altre repliche

Il sistema realizzato

- sistema Java con interfaccia a riga di comando
- esecuzione distribuita di entità client e server per la gestione dello stato del servizio, rappresentato da un valore intero
- comunicazione tra entità realizzata mediante Socket UDP

Funzionalità

Features incluse

- implementazione del protocollo in 3 fasi
- gestione dei checkpoint
- simulazione di Byzantine Fault nelle repliche
- processo di ViewChange e cambio di vista

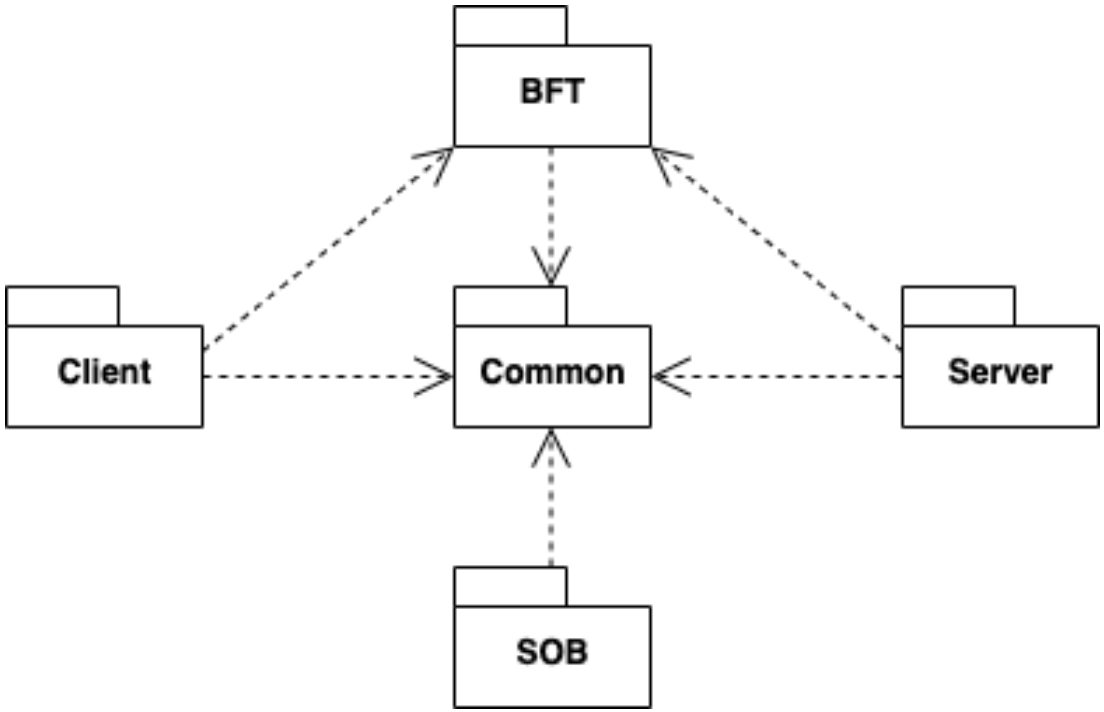
Features non incluse

- meccanismo di Proactive Recovery
- procedura di State Transfer and Checking
- procedura di Key Exchange

Struttura del sistema I

- **Common**: componente che offre utilità comuni a tutte le componente in gioco
- **PBFT**: componente che offre le funzionalità del protocollo PBFT a Client e Server
- **Client**: entità dedicata all'invio di richieste verso i Server per ottenere un risultato
- **Server**: entità con il compito di processare le richieste inviate dai Client, adottando politiche di consenso
- **SoB**: entità utile ad innescare eventi esterni al protocollo

Struttura del sistema II



Esecuzione del sistema I

```
$ gradle client:run --args=#
```

```
$ gradle server:run --args=#
```

```
$ gradle sob:run --args=#
```

Client	1	2	3	4	5	6	7
Porta	4450	4451	4452	4453	4454	4455	4456

Mappatura porte delle entità Client

Server	1	2	3	4	5	6	7
Porta	4470	4471	4472	4473	4474	4475	4476

Mappatura porte delle entità Server

Esecuzione del sistema II

- **client**: entità dipendente dall'immissione di comandi utente
  1. lettura delle configurazioni e inizializzazione delle socket
  2. invio di una richiesta
  3. attesa del risultato
     - 3.1 [se scade il timer] ritrasmissione della richiesta
- **server**: entità totalmente autonoma
  1. lettura delle configurazioni e inizializzazione delle socket
  2. ricezione di una richiesta
  3. scambio dei messaggi per il consenso o per ViewChange
     - 3.1 [ogni K esecuzioni] esecuzione procedura checkpoint
- **sob**: altra entità dipendente dall'immissione di comandi utente, invia messaggi senza attendere risposte
  1. lettura delle configurazioni e inizializzazione delle socket
  2. invio di un messaggio a client o server

Esecuzione del sistema III

Operazioni del client

	Forma breve	**Forma estesa**	**Risultato atteso**
Addizione	ADD *N*	ADDITION *N*	*STATO* + *N*
Sottrazione	SUB *N*	SUBTRACTION *N*	*STATO* - *N*
Moltiplicazione	MUL *N*	MULTIPLICATION *N*	*STATO* * *N*
Divisione	DIV *N*	DIVISION *N*	*STATO* / *N*
Lettura	READ	READ	*STATO*

N rappresenta l'operando dell'operazione richiesta

Esecuzione del sistema IV

Operazioni del SOB

	Forma breve	**Forma estesa**
Corruzione server	COR SER *N*	CORRUPT SERVER *N*
Riavvio server	RES SER *N*	RESTART SERVER *N*
Spegnimento server	SHU SER *N*	SHUTDOWN SERVER *N*
Riavvio client	RES CLI *N*	RESTART CLIENT *N*
Spegnimento client	SHU CLI *N*	SHUTDOUWN CLIENT *N*

N rappresenta l'identificativo della replica destinatario

Esecuzione del sistema V

Core operations

- l'esecuzione del sistema nelle operazioni *core* rispecchia fedelmente il funzionamento dell'algoritmo descritto
- la corruzione delle repliche da parte del SOB può avvenire in qualsiasi momento dell'esecuzione
  - ! ha un effetto osservabile solo sul *primary* che invierà messaggi con *seqnum* 0
  - !! innesca una ViewChange che permette di sostituire il *primary* corrotto
    - la replica corrotta ripristina il suo stato al termine della ViewChange

per un esempio completo di esecuzione *si rimanda alla relazione di progetto*

Proprietà fondamentali I

Safety

- le repliche sono d'accordo su un ordine totale e non effettuano mai commit di richieste distinte con lo stesso *seqnum*
- i *backup* corrotti non saranno mai in grado di avviare una VC ($f + 1$ repliche necessarie)
  - ! unica eccezione se la VC è avviata da un p corrotto
  - !! ma la normale operatività riprende al più dopo f processi di VC

Proprietà fondamentali II

Liveness

- le repliche effettuano un processo di VC nel caso non sia possibile eseguire una richiesta, se sospettano dell'attuale primary oppure alla scadenza di un timeout
- massimizzazione del tempo in cui $2f + 1$ repliche corrette, tra cui p, sono nella stessa view
  - ! la durata del timer raddoppia ogni volta che esso scade prima del termine della procedura

Proprietà fondamentali III

Fairness

- i client ottengono risposte anche quando sono presenti altri client che utilizzano il servizio.
- i *seqnum* sono assegnati con un criterio FIFO dal *primary*
- i *backup* mantengono le richieste in code FIFO
- durante le VC, i *backup* fermano il timer solo dopo aver eseguito una delle richieste in coda
  - ! il *primary* non può favorire l'esecuzione di alcune richieste a discapito di altre.