Consenso Distribuido

30221 - Sistemas Distribuidos

Unai Arronategui - Rafael Tolosana

Dpto. Informática e Ing. de Sistemas

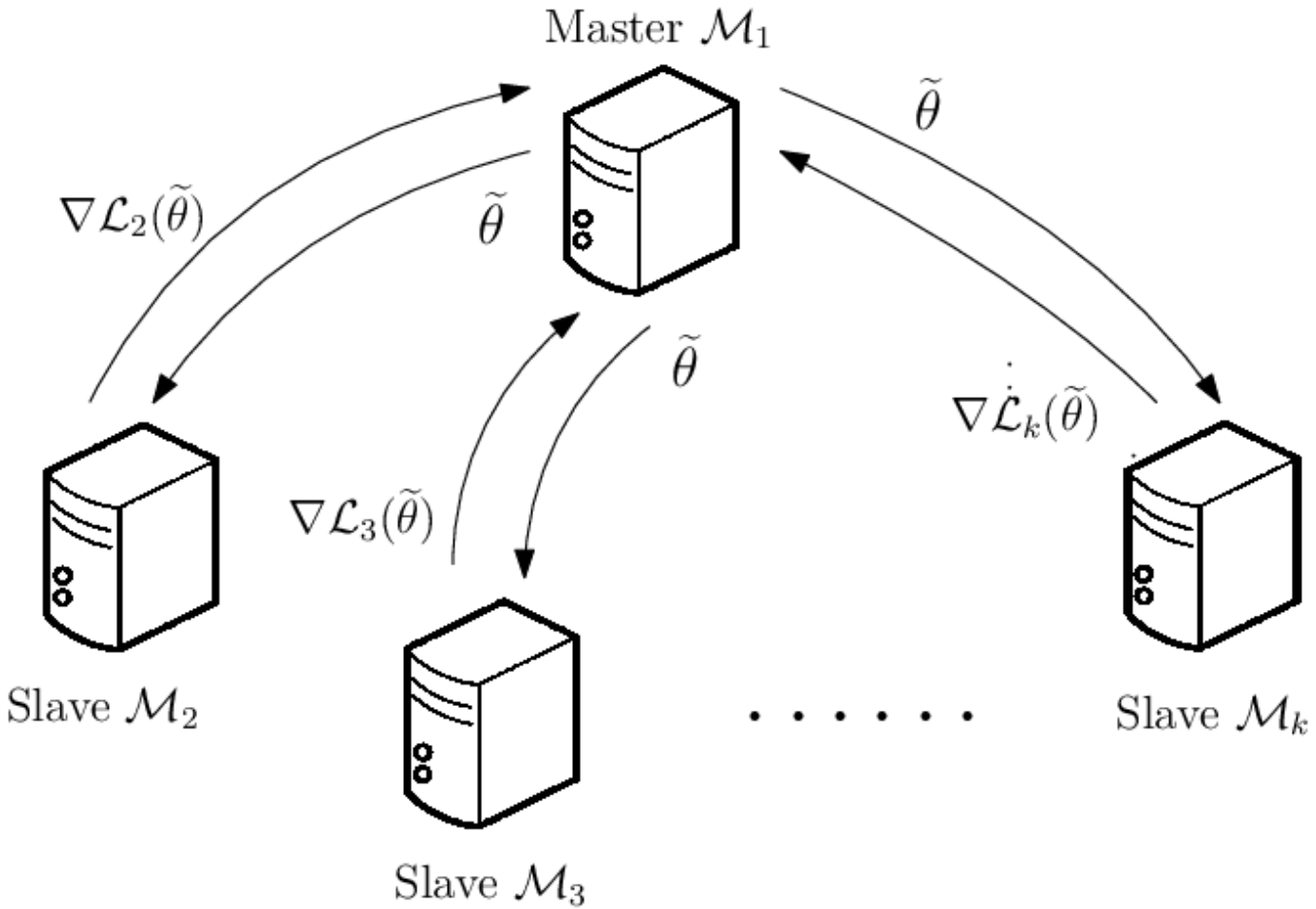
Lectura Recomendada

- Tanenbaum Van Steen, Distributed Systems: Principles and Paradigms, 3e, (c) 2017. **Section 8.2**

Motivación

Motivación

Aplicaciones con Estado: de CPU a Disco



Motivación

Dificultades Tolerancia a Fallos con Estado

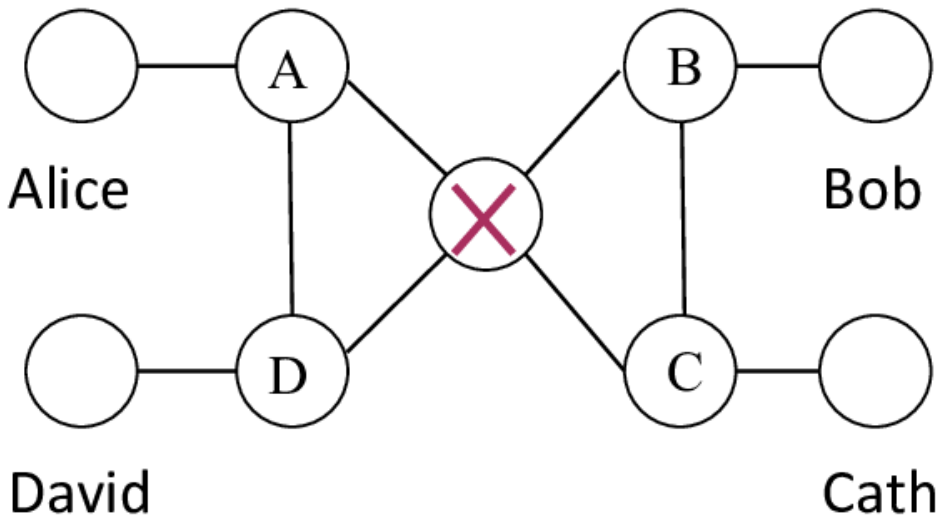
- Los fallos ya vistos

[1] Zhu, Zhenkai Bian, Chaoyi Afanasyev, Alexander Jacobson, Van Zhang, Lixia. (2012). Chronos: Serverless Multi-User Chat Over NDN.

Motivación

Dificultades Tolerancia a Fallos con Estado

- Los fallos ya vistos
- Partición de Red



[1]

[1] Zhu, Zhenkai Bian, Chaoyi Afanasyev, Alexander Jacobson, Van Zhang, Lixia. (2012). Chronos: Serverless Multi-User Chat Over NDN.

Motivación

Dos Propiedades de Concurrencia

- Corrección de la operación (*safety*):
  - **consistencia**

Motivación

Dos Propiedades de Concurrencia

- Corrección de la operación (*safety*):
  - **consistencia**
- Vivacidad (*liveness*):
  - **disponibilidad**

Motivación

Dos Propiedades de Concurrencia

- Corrección de la operación (*safety*):
  - **consistencia**
- Vivacidad (*liveness*):
  - **disponibilidad**

Una solución

- Combinar la **elección de líder** y **replicación**

Motivación

El Teorema CAP de Brewer [2]

Todo sistema de datos distribuido (estado) solo puede mantener 2 de las 3 propiedades siguientes simultáneamente:

[2] Brewer, Eric A. "Towards robust distributed systems." PODC. Vol. 7. No. 10.1145. 2000.

Motivación

El Teorema CAP de Brewer [2]

Todo sistema de datos distribuido (estado) solo puede mantener 2 de las 3 propiedades siguientes simultáneamente:

- **Consistencia** (**C**onsistencia)
  - Toda lectura recibe la escritura más reciente

[2] Brewer, Eric A. "Towards robust distributed systems." PODC. Vol. 7. No. 10.1145. 2000.

Motivación

El Teorema CAP de Brewer [2]

Todo sistema de datos distribuido (estado) solo puede mantener 2 de las 3 propiedades siguientes simultáneamente:

- **Consistencia** (**C**onsistencia)
  - Toda lectura recibe la escritura más reciente
- **Disponibilidad** (**A**vailability)
  - Toda petición recibe respuesta (consistente o no)

[2] Brewer, Eric A. "Towards robust distributed systems." PODC. Vol. 7. No. 10.1145. 2000.

Motivación

El Teorema CAP de Brewer [2]

Todo sistema de datos distribuido (estado) solo puede mantener 2 de las 3 propiedades siguientes simultáneamente:

- **Consistencia** (**C**onsistencia)
  - Toda lectura recibe la escritura más reciente
- **Disponibilidad** (**A**vailability)
  - Toda petición recibe respuesta (consistente o no)
- **Tolerante a Particiones de Red** (**P**artition tolerance)
  - El sistema continúa operando en presencia de un número arbitrario de mensajes que se pierden o se retrasan por la red

[2] Brewer, Eric A. "Towards robust distributed systems." PODC. Vol. 7. No. 10.1145. 2000.

Motivación

Dependiendo de la aplicación se puede poner énfasis en 2 propiedades:

- Consistencia y Disponibilidad
  - BBDD tradicionales
- Disponibilidad y Particionado
  - Bases de datos NOSQL, Cassandra...
- Consistencia y Particionado

Las técnicas de consenso tratan de cubrir las 3 propiedades: Consistencia, Disponibilidad y Particionado de Red

Consenso

Consenso

Definición

- Acuerdo entre los procesos de un sistema distribuido sobre un valor.

Consenso

Definición

- Acuerdo entre los procesos de un sistema distribuido sobre un valor.

Objetivos

- Consistencia
  - Mantener consistencia aún con nodos caídos
- Disponibilidad
  - Suprimir puntos de fallo único
- Tolerancia a particiones de red
  - Mantener consistencia y disponibilidad aunque sea en una sola partición de red
- No consideramos fallos bizantinos

Consenso

Dos aproximaciones al problema del consenso

- Simétrica, sin líder
  - Todos los procesos tienen el mismo rol
  - Un cliente puede contactar cualquier proceso (servidor)

Consenso

Dos aproximaciones al problema del consenso

- Simétrica, sin líder
  - Todos los procesos tienen el mismo rol
  - Un cliente puede contactar cualquier proceso (servidor)
- Asimétrica, con líder
  - En todo momento, hay un proceso líder, los demás acatan sus órdenes
  - Un cliente solo interactúa con el líder

Consenso

Propiedad de un Algoritmo de consenso

- **Acuerdo** (*agreement*)
  - La decisión de todos los procesos correctos de un grupo es la misma (mismo valor)
  - Procesos parados o con fallo no son considerados.

Consenso

Propiedad de un Algoritmo de consenso

- **Acuerdo** (*agreement*)
  - La decisión de todos los procesos correctos de un grupo es la misma (mismo valor)
  - Procesos parados o con fallo no son considerados.
- **Terminación**
  - todos los procesos correctos deben decidir sobre el valor en un tiempo finito

Consenso

Propiedad de un Algoritmo de consenso

- **Acuerdo** (*agreement*)
  - La decisión de todos los procesos correctos de un grupo es la misma (mismo valor)
  - Procesos parados o con fallo no son considerados.
- **Terminación**
  - todos los procesos correctos deben decidir sobre el valor en un tiempo finito
- **Integridad**
  - un proceso no puede modificar su decisión

Consenso

Propiedad de un Algoritmo de consenso

- **Acuerdo** (*agreement*)
  - La decisión de todos los procesos correctos de un grupo es la misma (mismo valor)
  - Procesos parados o con fallo no son considerados.
- **Terminación**
  - todos los procesos correctos deben decidir sobre el valor en un tiempo finito
- **Integridad**
  - un proceso no puede modificar su decisión
- **Validez**
  - Un valor decidido debe estar entre los valores propuestos a la decisión

Consenso

Modelo de Máquina de Estados Replicados

- Los algoritmos de consenso surgen en torno a las máquinas de estados replicados

Consenso

Modelo de Máquina de Estados Replicados

- Los algoritmos de consenso surgen en torno a las máquinas de estados replicados
- Cada estado representa una operación que se realiza

Consenso

Modelo de Máquina de Estados Replicados

- Los algoritmos de consenso surgen en torno a las máquinas de estados replicados
- Cada estado representa una operación que se realiza
- Todos los servidores procesan la misma operación en el mismo orden, partiendo desde el mismo estado inicial

Consenso

Modelo de Máquina de Estados Replicados

- Los algoritmos de consenso surgen en torno a las máquinas de estados replicados
- Cada estado representa una operación que se realiza
- Todos los servidores procesan la misma operación en el mismo orden, partiendo desde el mismo estado inicial
- Todas los servidores tienen una copia idéntica de la misma máquina de estados

Consenso

Modelo de Máquina de Estados Replicados

- Los algoritmos de consenso surgen en torno a las máquinas de estados replicados
- Cada estado representa una operación que se realiza
- Todos los servidores procesan la misma operación en el mismo orden, partiendo desde el mismo estado inicial
- Todas los servidores tienen una copia idéntica de la misma máquina de estados
- Este modelo proporciona una consistencia secuencial

Raft

Raft

Algoritmo de Consenso Raft

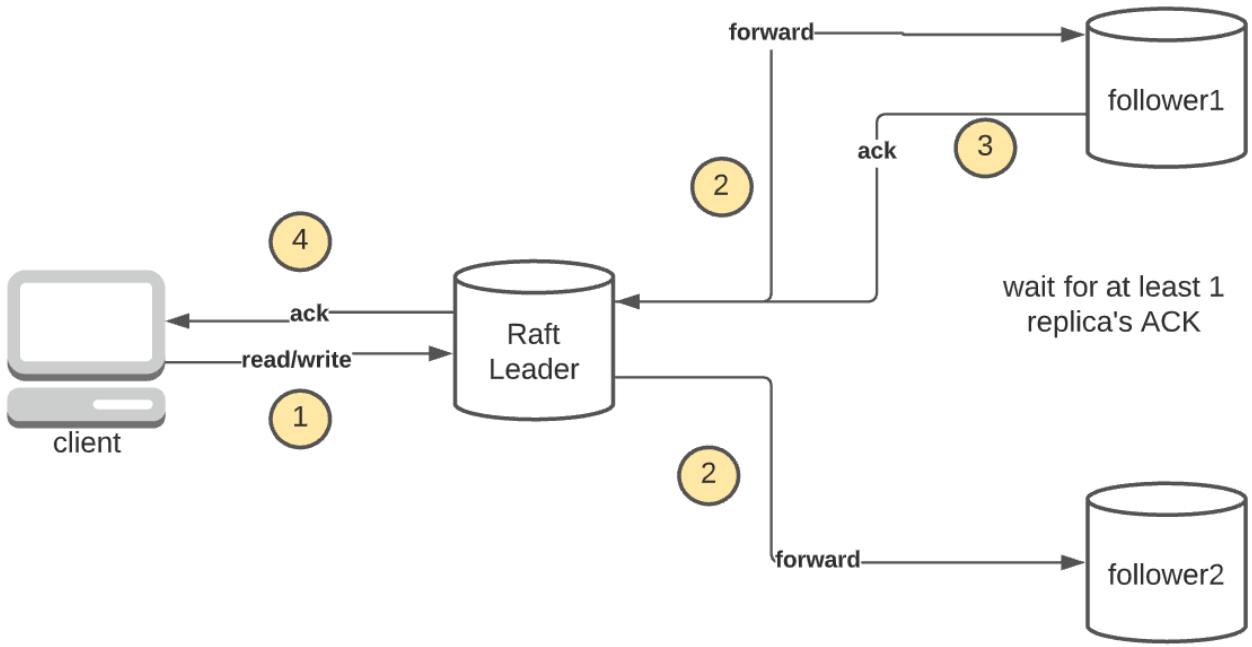
Componentes de Raft [3]

- **Elección de líder**
  - Se elige un nuevo líder cuando falla el líder actual
- **Operación normal (replicación de log)**
  - El líder recibe peticiones de los clientes y las registra localmente (escritura)
  - El líder replica su log a otros servidores
- **Corrección (safety)**
  - Corrección debida a la máquina de estados replicados
  - Si algún servidor ha registrado una operación en su log en un determinado índice, ningún otro servidor aplicará una operación distinta en dicho índice.

[3] Algoritmo Raft de Consenso, 2014 http://raftconsensus.github.io/

Raft

Flujo de datos en Raft en operación normal

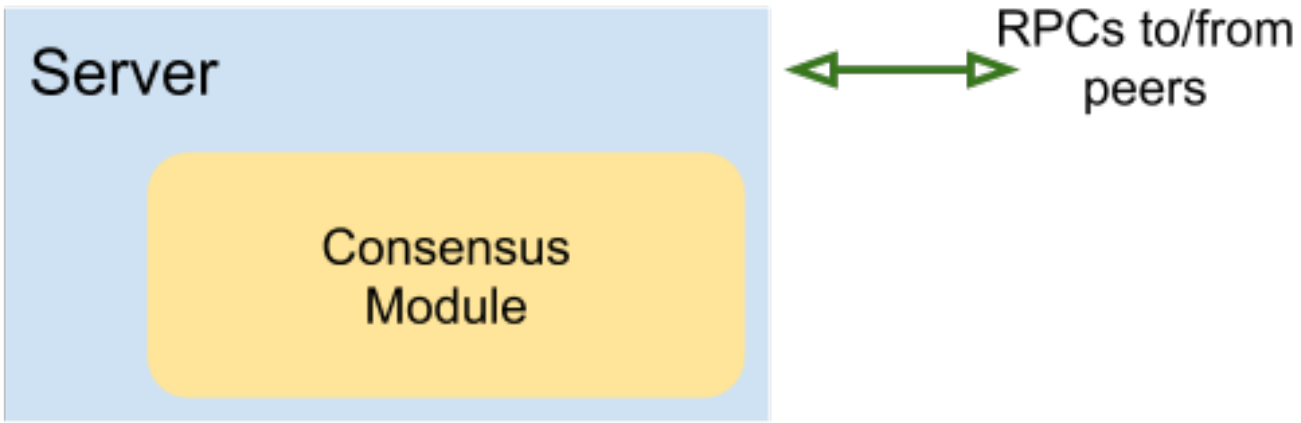


[4]

[4] https://towardsdatascience.com/raft-algorithm-explained-a7c856529f40

Raft

Arquitectura de Raft



[5]

Los servidores en Raft se comunican a través de RPC:

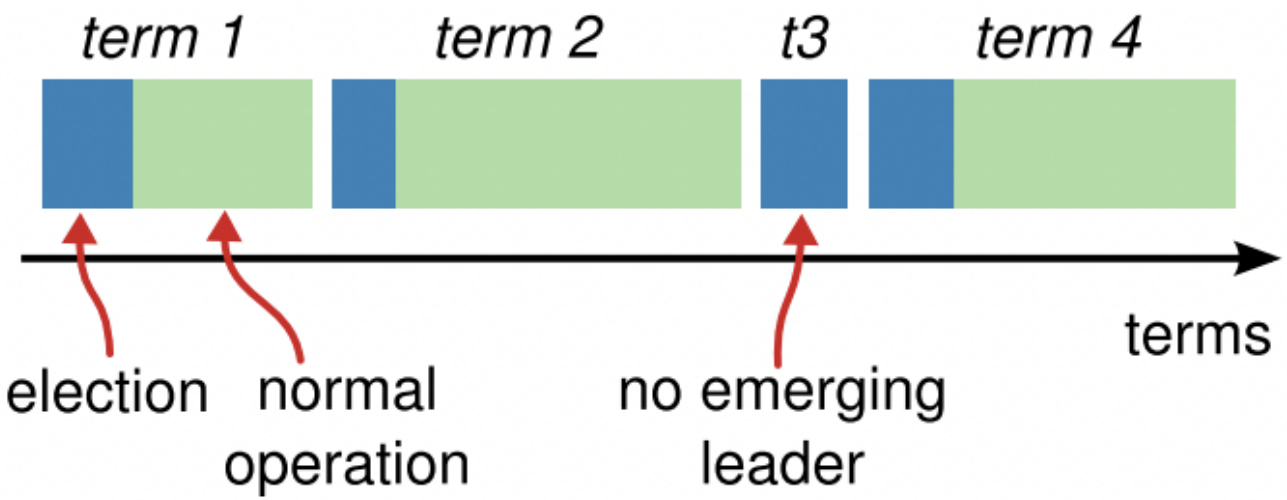
- procedimiento AppendEntries
- procedimiento RequestVote

[5] https://eli.thegreenplace.net/2020/implementing-raft-part-1-elections/

Raft

Mandatos (Terms)

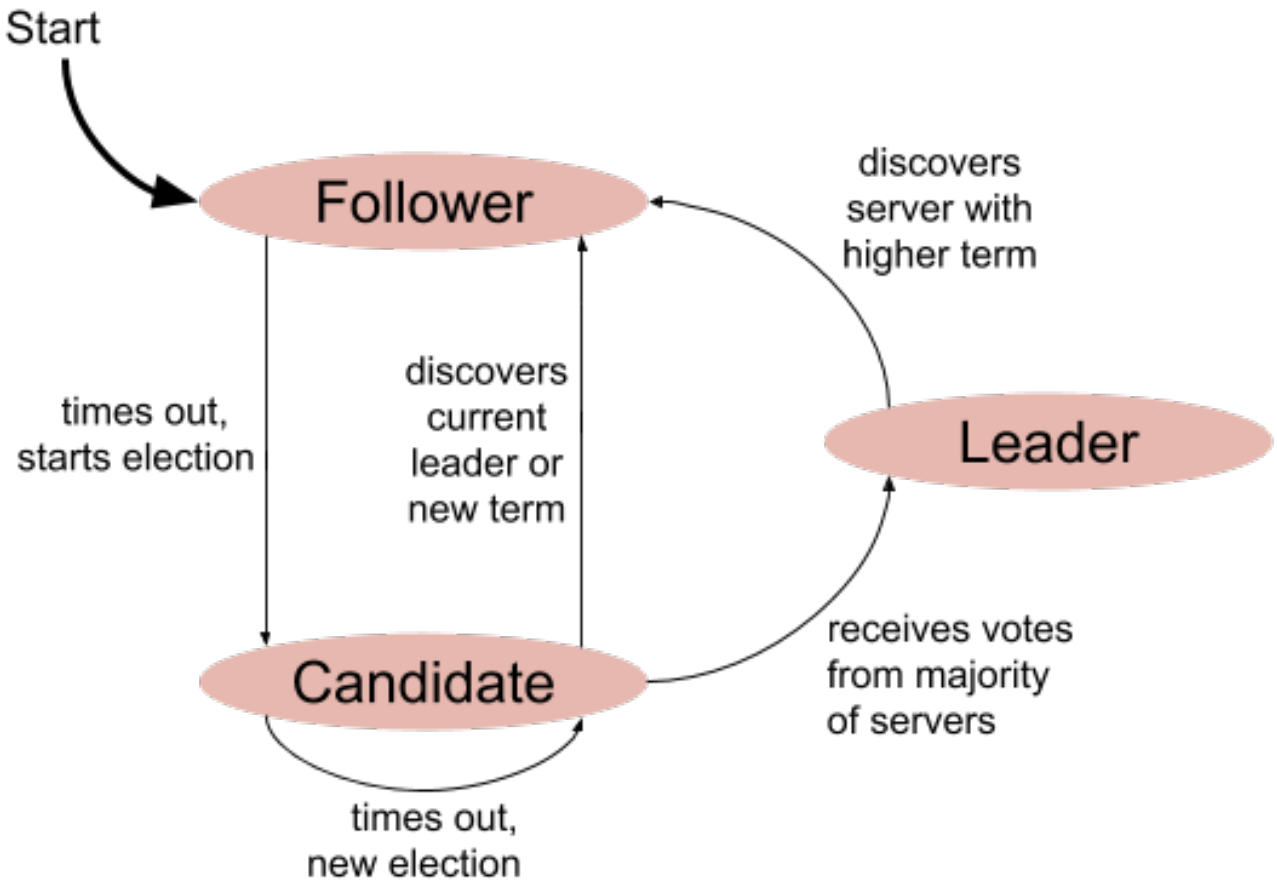
- Un mandato es un periodo de tiempo en que un servidor es el líder
  - tiene una fase de elección
  - puede tener o no una fase de operación normal
- Una nueva elección (tras un fallo) genera una nueva elección



[6] https://raft.github.io/

Ciclo de Vida en Raft

Ciclo de Vida de un Servidor Raft

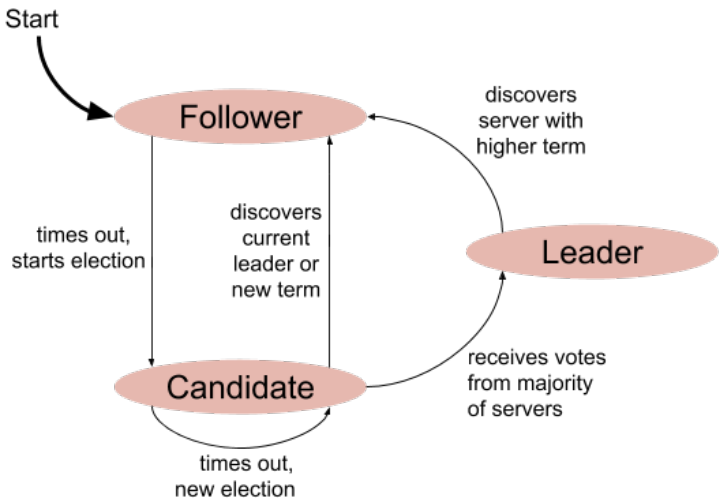


7

[7] https://eli.thegreenplace.net/2020/implementing-raft-part-1-elections/

Ciclo de Vida en Raft

Ciclo de Vida de un Servidor Raft



- Seguidor (*follower*): solo responde a instrucciones del líder (para replicación)
- Candidato (*candidate*): proceso que ha iniciado una nueva elección
- Líder (leader): proceso que gestiona peticiones de los clientes y las réplicas en los seguidores

Ciclo de Vida en Raft

- Los **servidores** comienzan como seguidores
- Esperan recibir latidos desde líder o desde candidatos.
- Líder debe enviar latidos para mantener autoridad
- Si tiempo de expiración de elección transcurre sin latidos:
  - Seguidor asume que líder ha caído
  - Seguidor empieza nueva elección
  - Tiempos de expiración **aleatorios** y típicos entre 100 y 500 ms

Temporizador en Raft

Temporizador (timer)

- Elemento fundamental en Raft

Temporizador en Raft

Temporizador (timer)

- Elemento fundamental en Raft
- El líder envía un latido (*heartbeat*) periódicamente a todos los seguidores

Temporizador en Raft

Temporizador (timer)

- Elemento fundamental en Raft
- El líder envía un latido (*heartbeat*) periódicamente a todos los seguidores
- Cada seguidor tiene un temporizador que se desactiva cada vez que se recibe un mensaje del líder

Temporizador en Raft

Temporizador (timer)

- Elemento fundamental en Raft
- El líder envía un latido (*heartbeat*) periódicamente a todos los seguidores
- Cada seguidor tiene un temporizador que se desactiva cada vez que se recibe un mensaje del líder
- Si los mensajes del líder no llegan, expira el temporizador y el **seguidor inicia una elección** (cambia a candidato).

Temporizador en Raft

El **Líder** utiliza el procedimiento RPC AppendEntries vacío, como latido

- Realiza invocaciones periódicas a todos los seguidores
- AppendEntries incluye:
  - el mandato del líder (id)
  - Además de determinada información relacionada con la consistencia

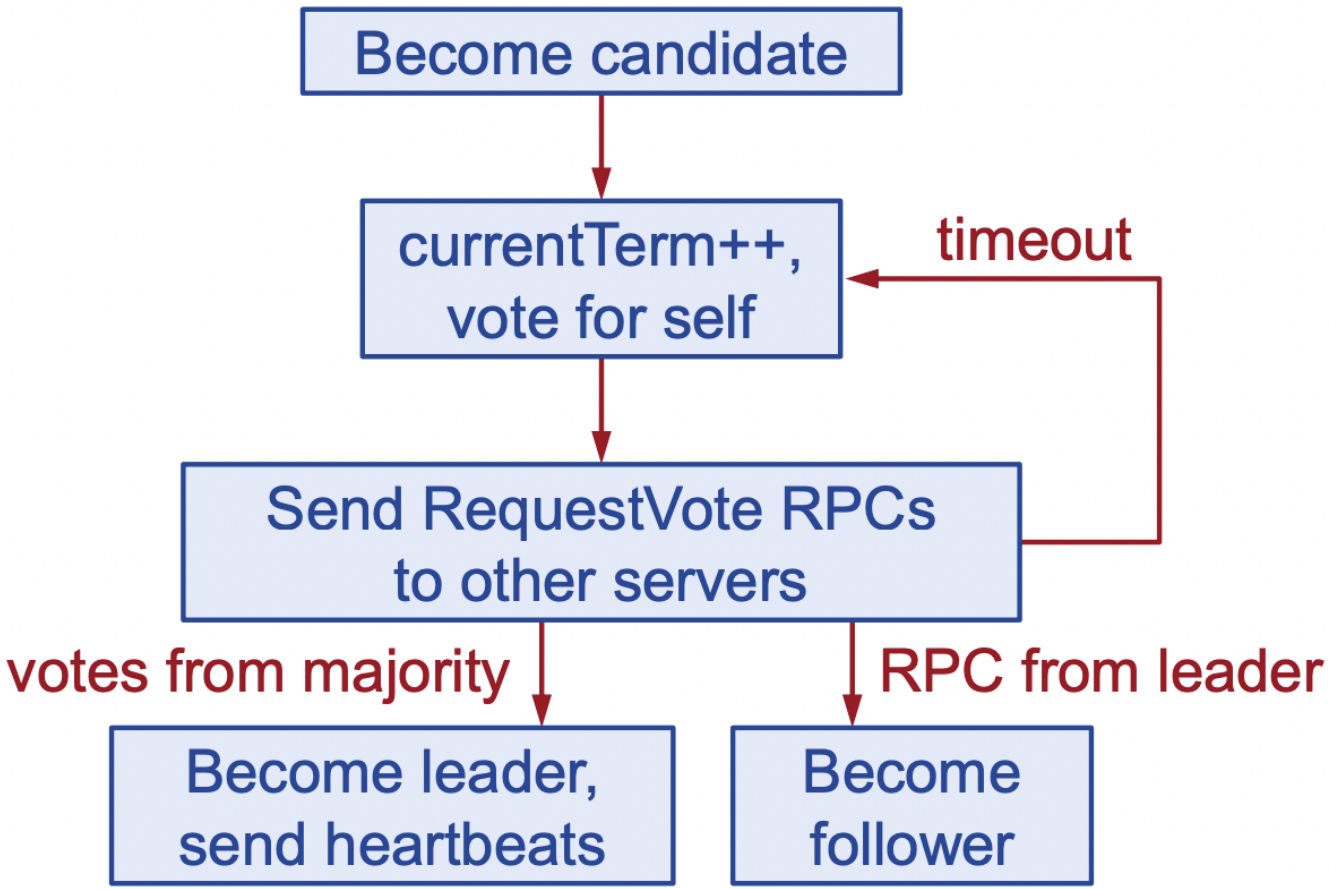
Temporizador en Raft

Temporizador en el seguidor del Alg de Raft

fragmento en Golang

```
for nr.rol == FOLLOWER {
    select {
        case <- nr.canalHeartbeat:
            // Si recibe latido, continúa siendo FOLLOWER
            // Puede ser de un líder o de un candidato al que le hemos
            // dado el voto
        case <- time.After(getRandomTimeout(nr)):
            // Si se termina el timeout de latido, se convierte en CANDIDATE
            nr.rol = CANDIDATE
    }
}
```

Elección de Líder en Raft



8

[8] taken by Professor John Ousterhout's slides

Elección de Líder en Raft

Proceso de Elección de un seguidor

1. Incrementar el mandato actual del seguidor
2. Cambiar el estado del seguidor a candidato
3. Votar por sí mismo
4. Enviar PeticionVoto a todos los demás servidores hasta que:
   - Recibe votos de una **mayoría** simple de servidores
     - Se convierte en líder
     - Envía latidos a todos los demás (notificación)
   - Recibe **latido de un líder**
     - Vuelve al estado **seguidor**
   - **Nadie gana** la elección (empate o expira el temporizador)
     - Inicia una **nueva elección, goto 1**

Elección de Líder en Raft

Cuando un proceso recibe una petición de voto

- Un proceso solo puede votar una vez en un mandato

Elección de Líder en Raft

Cuando un proceso recibe una petición de voto

- Un proceso solo puede votar una vez en un mandato
- Se vota al primer candidato que lo solicite

Elección de Líder en Raft

Cuando un proceso recibe una petición de voto

- Un proceso solo puede votar una vez en un mandato
- Se vota al primer candidato que lo solicite
- Ante una petición de voto pueden darse estas opciones:
  - Si el mandato recibido del candidato es menor que el mandato del seguidor, se deniega el voto
  - Si se ha votado ya a otro candidato en ese mandato, se deniega el voto
  - Se da el voto si no se ha votado a nadie para ese mandato.

Elección de Líder en Raft

Cuando un proceso recibe una petición de voto

- Un proceso solo puede votar una vez en un mandato
- Se vota al primer candidato que lo solicite
- Ante una petición de voto pueden darse estas opciones:
  - Si el mandato recibido del candidato es menor que el mandato del seguidor, se deniega el voto
  - Si se ha votado ya a otro candidato en ese mandato, se deniega el voto
  - Se da el voto si no se ha votado a nadie para ese mandato.

Dos candidatos diferentes no pueden acumular mayorías en un mismo mandato

Replicación del Registro

Registro (*log*)

- **Estructura**: entrada = (índice, mandato, operación)

Replicación del Registro

Registro (*log*)

- **Estructura**: entrada = (índice, mandato, operación)
- El registro se guarda en almacenamiento persistente

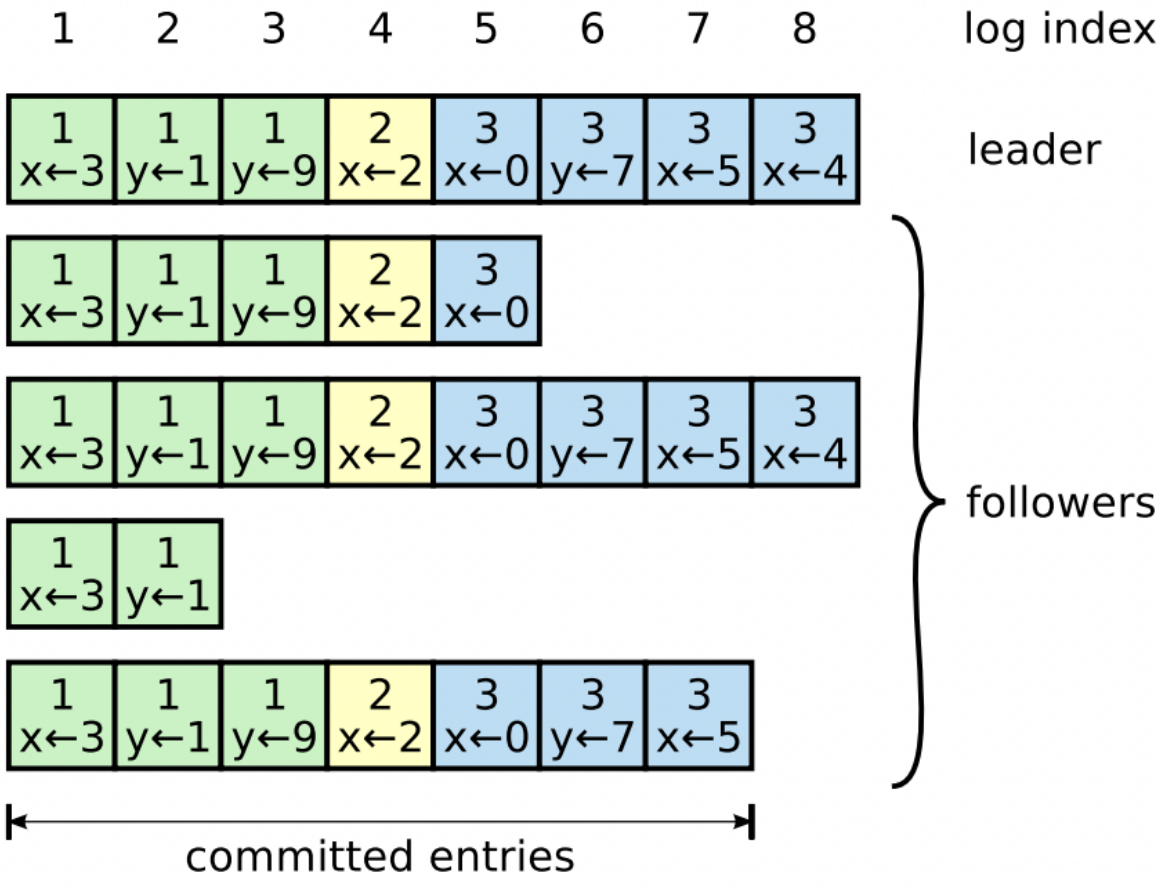
Replicación del Registro

Registro (*log*)

- **Estructura**: entrada = (índice, mandato, operación)
- El registro se guarda en almacenamiento persistente
- Una entrada en el registro está *comprometida* (*committed*) si:
  - Se ha almacenado por una mayoría de servidores

Replicación del Registro

Ejemplo de un registro

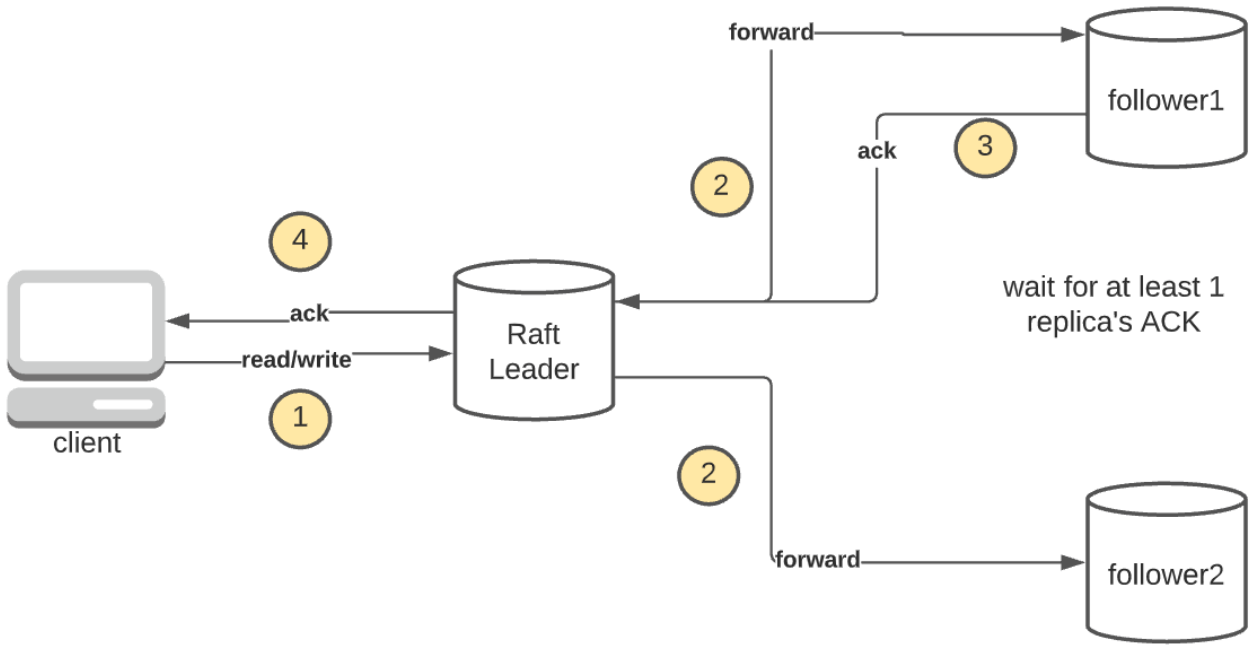


9

[9] https://raft.github.io/raft.pdf

Replicación del Registro

Flujo de datos en Raft en operación normal



10

[10] https://towardsdatascience.com/raft-algorithm-explained-a7c856529f40

Replicación del Registro

Operativa Normal (sin fallos)

1. Un cliente envía una petición (operación) al líder

Replicación del Registro

Operativa Normal (sin fallos)

1. Un cliente envía una petición (operación) al líder
2. El líder añade operación en su registro

Replicación del Registro

Operativa Normal (sin fallos)

1. Un cliente envía una petición (operación) al líder
2. El líder añade operación en su registro
3. El líder envia RPC AñadirEntradas a los seguidores (pueden ser varias operaciones simultáneas por RPC)

Replicación del Registro

Operativa Normal (sin fallos)

1. Un cliente envía una petición (operación) al líder
2. El líder añade operación en su registro
3. El líder envia RPC AñadirEntradas a los seguidores (pueden ser varias operaciones simultáneas por RPC)
4. Una vez la(s) entrada(s) comprometida(s) porque ha(n) obtenido mayoría:
   - El líder pasa operación a su máquina de estados y devuelve el resultado al cliente
   - El líder notifica a seguidores de las entradas comprometidas en subsiguientes RPCs AñadirEntradas
   - Los seguidores pasan entradas comprometidas a sus máquinas de estados

Replicación del Registro

¿Seguidores caídos o lentos?

- El líder reintenta RPCs AñadirEntradas hasta que logra compromiso de mayoría
- El líder reintenta RPCs AñadirEntradas hasta que todos los seguidores tienen el mismo registro

En Operativa Normal todos los registros de todos los servidores son consistentes
Un cambio de líder puede conllevar inconsistencias

- El líder antiguo no ha podido replicar sus entradas en el registro de los seguidores

Replicación del Registro

Consistencia del Registro (*Log Matching Property*)

- Si entradas de registro en diferentes servidores tienen el **mismo índice y mandato**:
  - Almacenan la misma operación

[11] http://raft.github.io

Replicación del Registro

Consistencia del Registro (*Log Matching Property*)

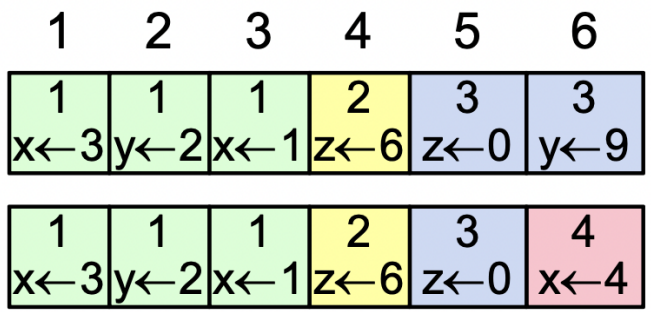
- Si entradas de registro en diferentes servidores tienen el **mismo índice y mandato**:
  - Almacenan la misma operación
  - Los registros son idénticos en todas la entradas anteriores

[11] http://raft.github.io

Replicación del Registro

Consistencia del Registro (*Log Matching Property*)

- Si entradas de registro en diferentes servidores tienen el **mismo índice y mandato**:
  - Almacenan la misma operación
  - Los registros son idénticos en todas la entradas anteriores
- Si una determinada entrada está comprometida, **todas las entradas anteriores** también están comprometidas

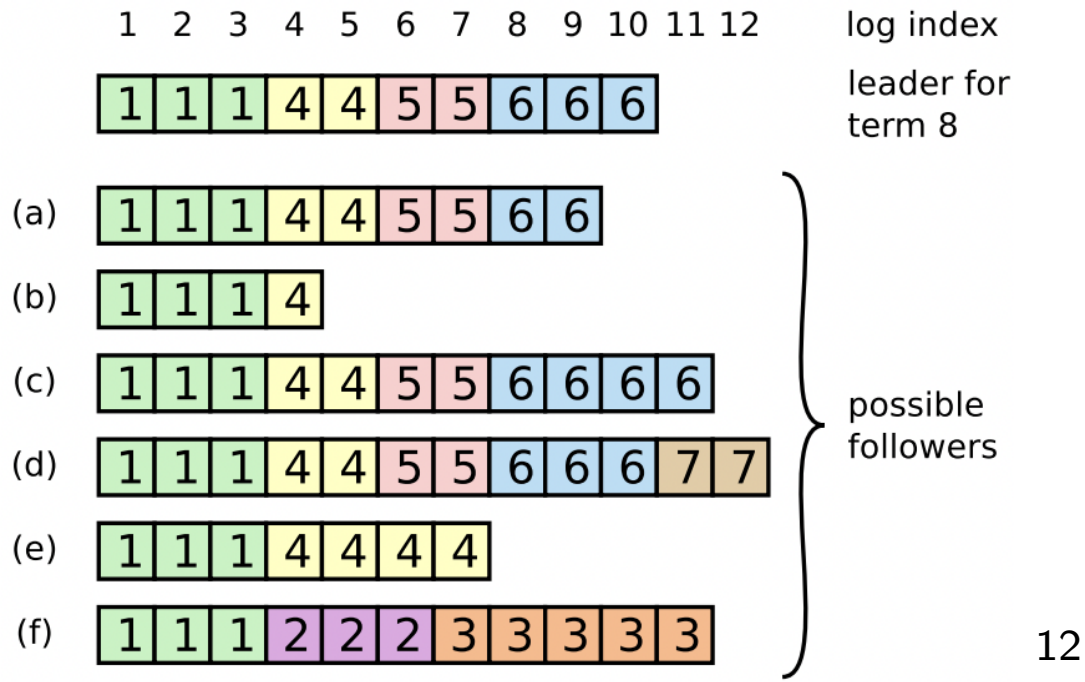


[11]

[11] http://raft.github.io

Replicación del Registro

Escenarios posibles tras cambio de líder

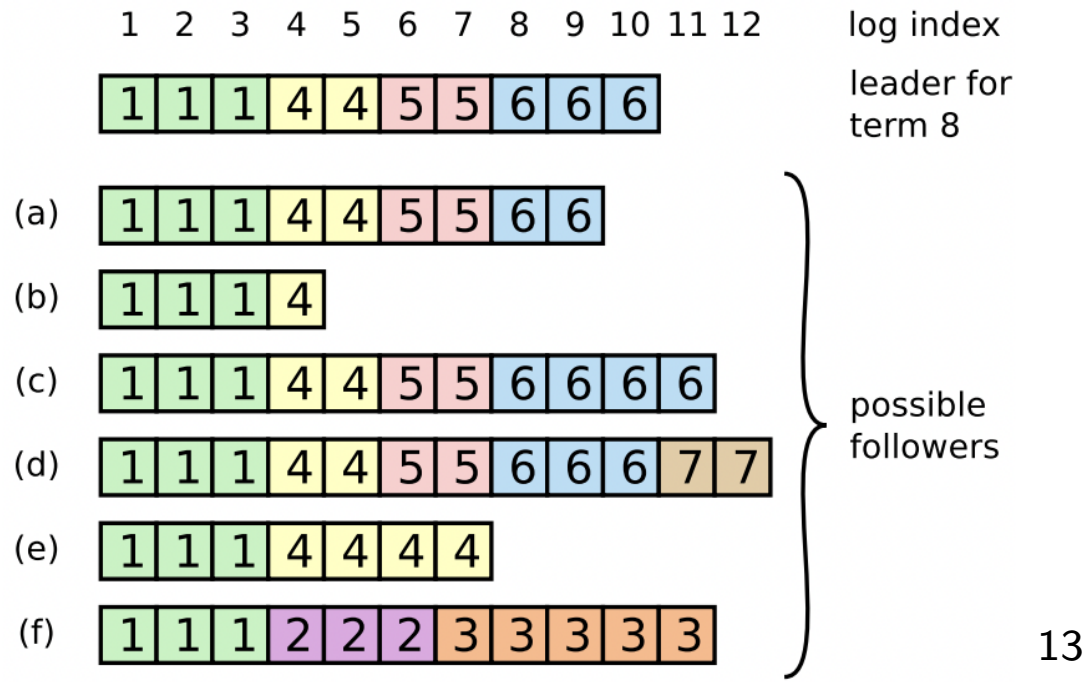


12

[12] http://raft.github.io

Replicación del Registro

Escenarios posibles tras cambio de líder



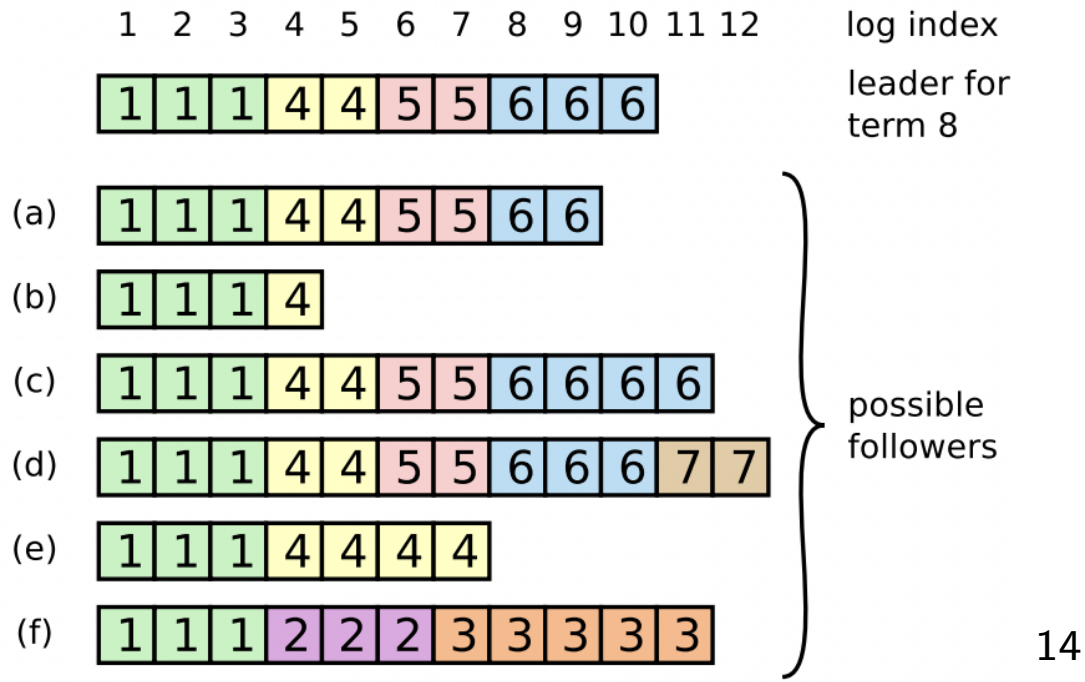
13

a-b entradas ausentes respecto del líder

[13] http://raft.github.io

Replicación del Registro

Escenarios posibles tras cambio de líder



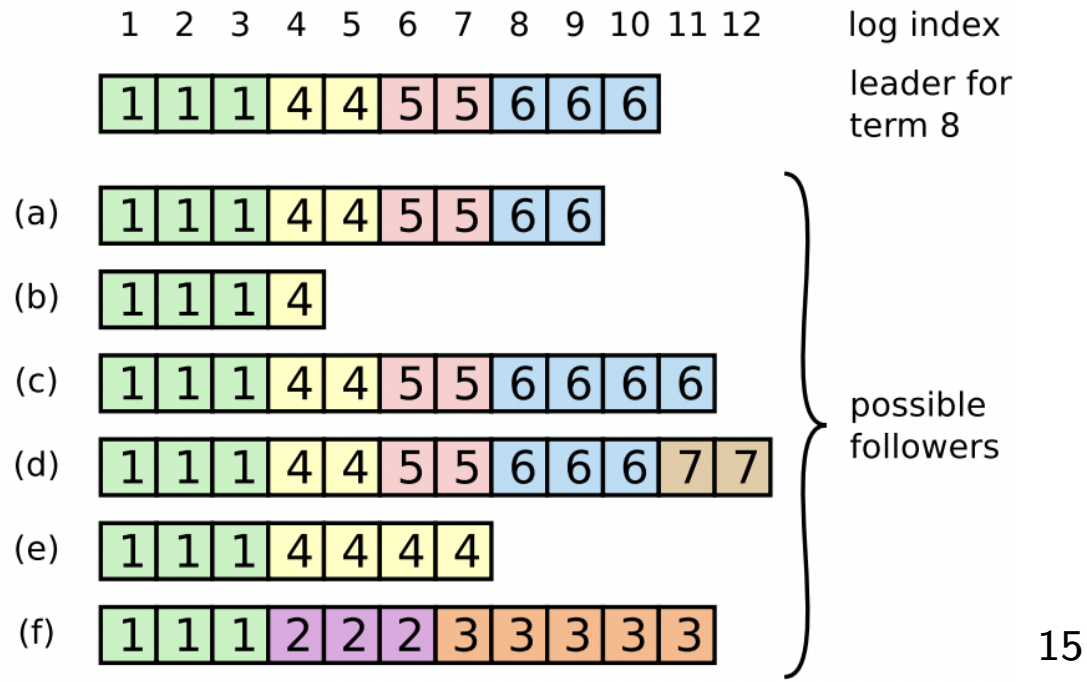
14

c-d entradas irrelevantes respecto del líder

[14] http://raft.github.io

Replicación del Registro

Escenarios posibles tras cambio de líder



15

e-f ausentes e irrelevantes

[15] http://raft.github.io

Replicación del Registro

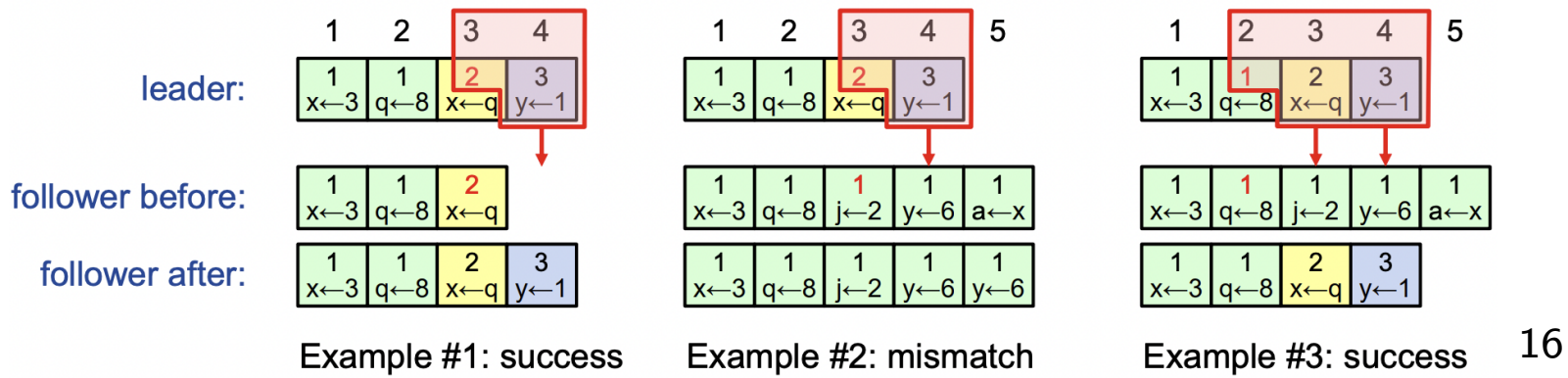
Cambio de líder: comprobación de corrección de entradas

- Un líder antiguo puede haber dejado inconsistencias
- **No** hay procedimiento especial para el nuevo líder
- Al interactuar con los seguidores mediante RPC AñadirEntrada, se detectarán inconsistencias

Replicación del Registro

Append Entries: Comprobación de Consistencia

- AppendEntries incluye <**índice, mandato**> de la última entrada
- El seguidor tiene que tener una entrada que coincida, o la rechaza
- Se asegura consistencia mediante pasos de inducción



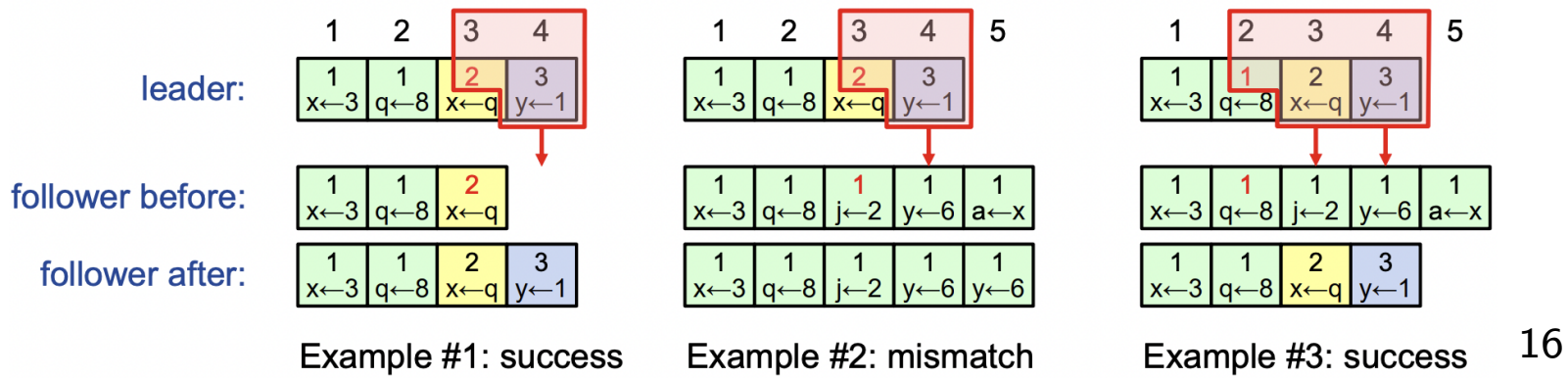
[16]

[16] http://raft.github.io

Replicación del Registro

Append Entries: Comprobación de Consistencia

- AppendEntries incluye <**índice, mandato**> de la última entrada
- El seguidor tiene que tener una entrada que coincida, o la rechaza
- Se asegura consistencia mediante pasos de inducción



[16]

El líder guarda **índice siguiente** para cada servidor

[16] http://raft.github.io

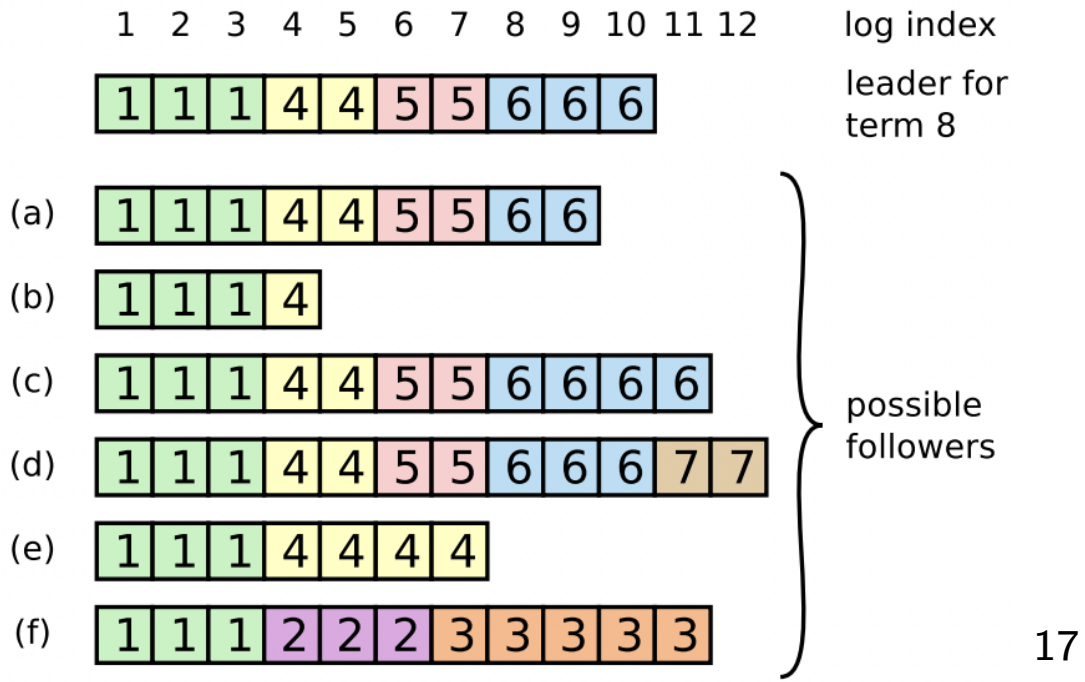
Replicación del Registro

Cambio de líder

- El líder sobreescribirá el registro del seguidor con inconsistencias
- Eventualmente, conseguirá que registros de seguidores sean idénticos a los del líder
- Múltiples caídas pueden dejar entradas todavía no comprometidas y, en algunos casos, no válidas (no ha habido mayoría)

Corrección de Inconsistencias

Cuidado... hace falta algo más



17

[17] http://raft.github.io

Corrección de Inconsistencias

Requisitos de Corrección

- Una vez que una entrada de registro ha sido comprometida (por mayoría), ninguna otra máquina de estados debe aplicar un valor diferente para esa entrada de registro.
- **Propiedad de correción de Raft:**
  - Si un líder ha decidido que una entrada de registro está comprometida, esa entrada estará presente en todos los registros de todos los líderes futuros.
- **Esta propiedad garantiza req de protección**
  - Los líderes nunca sobreescriben entradas en sus registros, solo entradas en el registro del líder pueden comprometerse.
  - Las entradas deben comprometerse antes de aplicarlas a máquina de estados

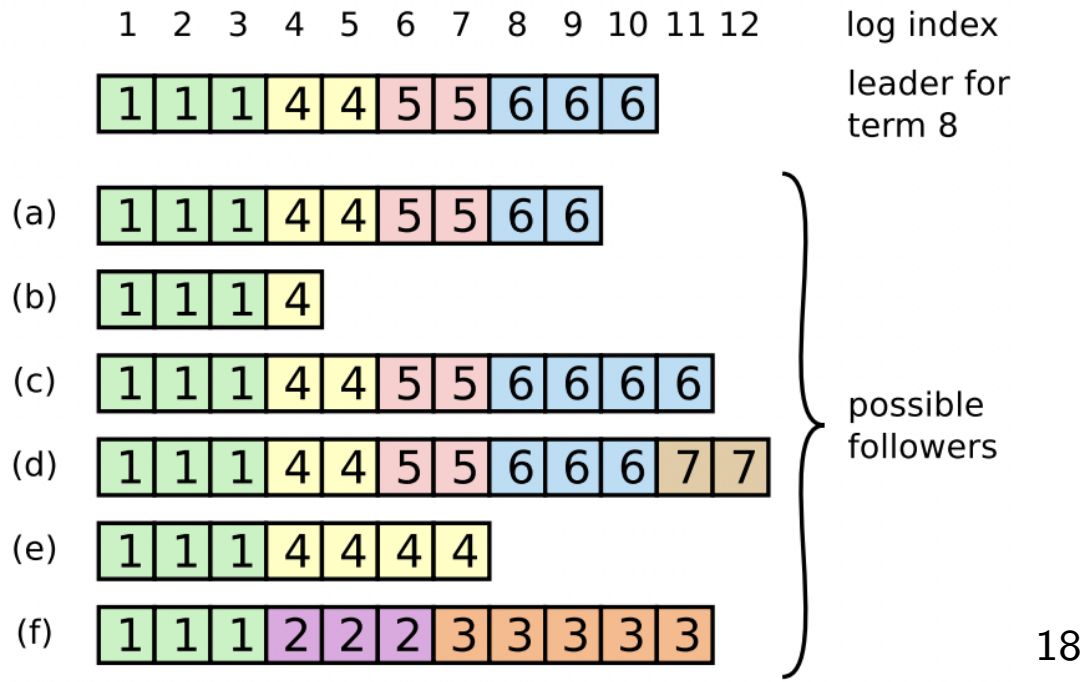
Corrección de Inconsistencias

Seleccionar al mejor líder

- Durante las elecciones, **elegir candidatos con más posibilidad de contener todas las entradas comprometidas**
  - No utilizar sólo modo simple por mayorías. Candidatos incluyen información de registro (indice y mandato de última entrada de registro) en mensaje de solicitud de voto
  - Utilización de métodos elección con rangos y mayoría
- Un Servidor S1 niega su voto y liderazgo a candidato C1 si registro S1 está más completo:
  - Esto permite seleccionar al líder con registro más completo

Corrección de Inconsistencias

El servidor f no puede ser líder

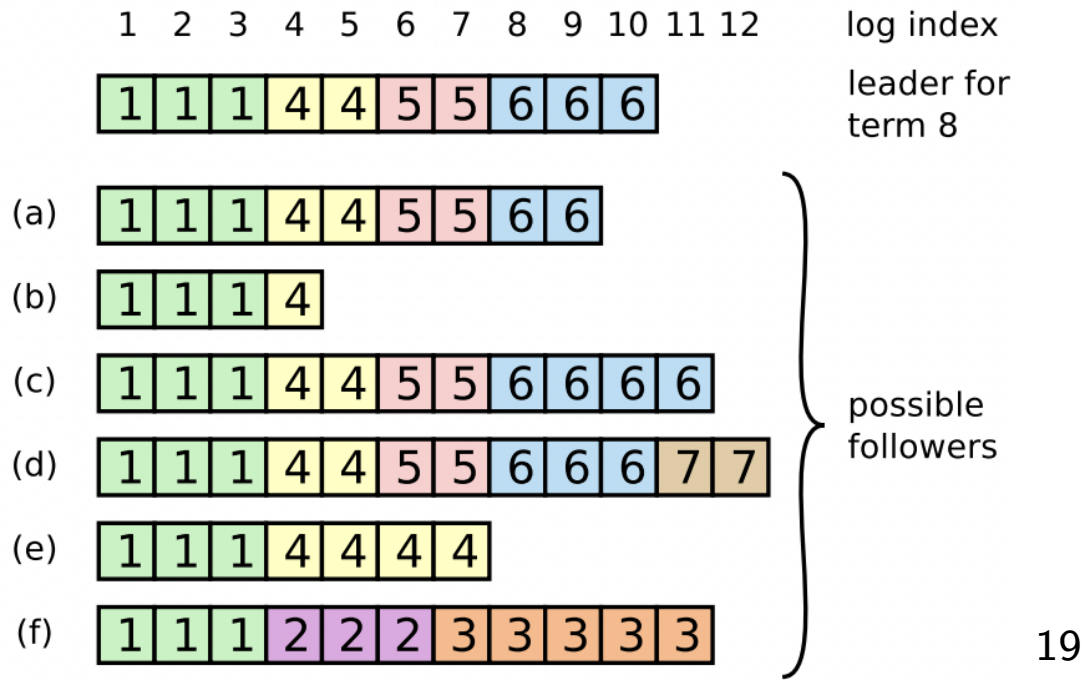


18

[18] http://raft.github.io

Corrección de Inconsistencias

¿Y los demás?



19

[19] http://raft.github.io

Corrección de Inconsistencias

Neutralizar Líderes Antiguos

- Los mandatos sirven para neutralizar líderes antiguos

Corrección de Inconsistencias

Neutralizar Líderes Antiguos

- Los mandatos sirven para neutralizar líderes antiguos
- También para neutralizar candidatos
  - En el RPC va el mandato del emisor

Corrección de Inconsistencias

Neutralizar Líderes Antiguos

- Los mandatos sirven para neutralizar líderes antiguos
- También para neutralizar candidatos
  - En el RPC va el mandato del emisor
  - Si alguien se detecta desfasado, pasa a seguidor automáticamente

Corrección de Inconsistencias

Reparación de Registros de Entradas sin comprometer

- Un nuevo líder podría intentar comprometer entradas de otros mandatos previos...

Corrección de Inconsistencias

Reparación de Registros de Entradas sin comprometer

- Un nuevo líder podría intentar comprometer entradas de otros mandatos previos...
- Raft no lo permite:
  - Un líder solo cuenta entradas registradas de su mandato

Corrección de Inconsistencias

Reparación de Registros de Entradas sin comprometer

- Un nuevo líder podría intentar comprometer entradas de otros mandatos previos...
- Raft no lo permite:
  - Un líder solo cuenta entradas registradas de su mandato
  - Una vez que una entrada del mandato actual se comprometa (contando mayoría), entonces las entradas previas se comprometerán indirectamente (Log Matching Property).

El cliente de Raft

¿Qué hace un cliente ante la caída de un líder?

El cliente de Raft

¿Qué hace un cliente ante la caída de un líder?

- Contactar con algún servidor cualquiera
  - Si no es el líder, este indica quién es

El cliente de Raft

¿Qué hace un cliente ante la caída de un líder?

- Contactar con algún servidor cualquiera
  - Si no es el líder, este indica quién es

El líder no responde al cliente hasta que la entrada se ha comprometido

El cliente de Raft

¿Qué sucede si un líder cae tras comprometer la entrada, pero antes de responder al cliente?
¿El cliente reintenta una operación?

El cliente de Raft

¿Qué sucede si un líder cae tras comprometer la entrada, pero antes de responder al cliente?
¿El cliente reintenta una operación?
Solución

- Cliente incluye un ID único en cada comando
- Servidor incluye ID en entrada de registro
- Antes de aceptar un comando, líder comprueba si tiene entrada con ese ID en su registro
- Si ID encontrado, ignora nuevo comando y devuelve respuesta de comando anterior

El cliente de Raft

¿Qué sucede si un líder cae tras comprometer la entrada, pero antes de responder al cliente?
¿El cliente reintenta una operación?
Solución

- Cliente incluye un ID único en cada comando
- Servidor incluye ID en entrada de registro
- Antes de aceptar un comando, líder comprueba si tiene entrada con ese ID en su registro
- Si ID encontrado, ignora nuevo comando y devuelve respuesta de comando anterior

Resultado : semántica de "exactemente uno"

Cambios de Configuración

Raft permite cambiar dinámicamente el número de réplicas

- Razones de mantenimiento

Cambios de Configuración

Raft permite cambiar dinámicamente el número de réplicas

- Razones de mantenimiento
- Cambio en el grado de replicación
- Sin necesidad de parar máquinas

Cambios de Configuración

No se puede conmutar directamente porque puede haber conflictos con las mayorías

- La idea es utilizar una fase en que conviven mayoría antigua y mayoría nueva

Temporizaciones y Disponibilidad

- **La corrección no depende del tiempo**

Temporizaciones y Disponibilidad

- **La corrección no depende del tiempo**
- Pero el tiempo afecta a la disponibilidad

Temporizaciones y Disponibilidad

- **La corrección no depende del tiempo**
- Pero el tiempo afecta a la disponibilidad
- Elección de líder, crítico en disponibilidad
  - multicast $<$ timeout expiración líder $<$ MTBF
- multicast, envío RPC: acceso a disco (0,5 - 20 ms)
- timeout exp líder: 20ms - 500 ms
- MTBF: meses

Consenso Distribuido

30221 - Sistemas Distribuidos

Unai Arronategui - Rafael Tolosana

Dpto. Informática e Ing. de Sistemas