ZooKeeper

El Servicio ZooKeeper

ZooKeeper es un servicio para coordinar procesos de aplicaciones distribuidas.

Terminología:

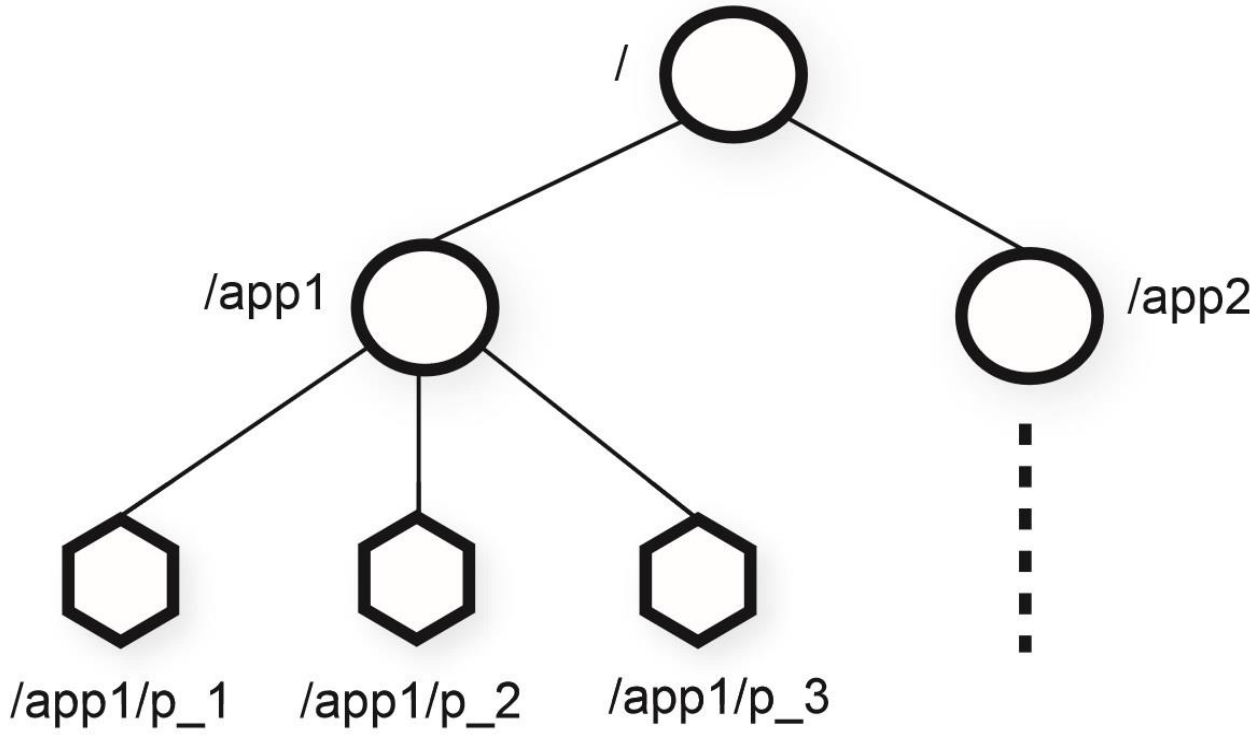
- Cliente: usuario del servicio ZooKeeper.
- Servidor: un proceso proporcionado por el servicio ZooKeeper.
- Znodo: un nodo de datos en memoria en los datos del ZooKeeper.
- Data Tree: un espacio de nombres jerárquico en el que se organizan los znodes.
- Sesión: generado por los clientes al conectarse al ZooKeeper . Los clientes obtienen un manejador de sesión a través del cual emiten sus solicitudes.

Visión General del Servicio

- ZooKeeper provee a sus clientes una abstracción de un conjunto de nodos de datos (znodos) organizados acorde a un espacio de nombres jerárquico.
- Los znodes son objetos de datos que los clientes manipulan a través de la API de ZooKeeper.

Tipos de znodes:

- Regular: los clientes manipulan estos nodos creándolos y eliminándolos explícitamente.
- Efímeros: creados por los clientes. Son eliminados explícitamente por los clientes o por el sistema cuando la sesión que creo el znodo termina.



- Ilustración del espacio de nombres jerárquico de ZooKeeper

- Los znodos efímeros no pueden tener hijos en el data tree.
- Los clientes pueden fijar una **bandera secuencial** cuando crean un nuevo znodo. Los znodos creados con una bandera secuencial tienen el valor de un contador monótonamente creciente agregado a su nombre: Si p es el padre del nuevo nodo n, el valor secuencia de n nunca será menor que el valor en el nombre de cualquier otro znodo secuencial creado debajo de p.
- ZooKeeper implementa **vigilancias (watch)** para permitir a los clientes recibir notificaciones oportunas acerca de cambios. Cuando un cliente solicita una operación de lectura con una bandera de vigilancia fijada, el servidor promete al cliente notificarle cuando la información retornada haya cambiado.

Modelo de Datos

- Esencialmente es un sistema de archivos con una API simplificada y sólo lecturas y escrituras de datos, o una tabla key/value con keys jerárquicas.
- Los znodos no están diseñados para almacenamiento de datos en general, en lugar de ello mapean a abstracciones de las aplicaciones del cliente, por lo general corresponden a meta-data usada para fines de coordinación.
- ZooKeeper permite que los clientes almacenen en los znodos información que pueda ser usada para meta-datos o configuración en computación distribuída.

Sesiones

- Los clientes que se conectan al ZooKeeper inician una sesión.
- Las sesiones tienen un timeout (tiempo muerto) asociado.
- ZooKeeper considera al cliente como defectuoso si no recibe algo de su sesión por un tiempo mayor que el timeout.
- Una sesión termina cuando el cliente la cierra explícitamente o cuando ZooKeeper detecta que el cliente es defectuoso.
- Dentro de la sesión, el cliente puede observar una sucesión de cambios de estado que reflejan la ejecución de sus operaciones.
- Las sesiones habilitan que los clientes puedan moverse transparentemente de un servidor a otro dentro del conjunto ZooKeeper.

API del Cliente

- **create(path, data, flags)**: crea un znodo de nombre path, almacena data[] y retorna el nombre del nuevo znodo. flags habilita al cliente a escoger el tipo de znodo a crear: regular o efímero, y fijar una bandera secuencial.
- **delete(path, version)**: elimina el znodo path si dicho nodo está en la versión esperada.
- **exist(path, watch)**: retorna true si existe un znodo de nombre de ruta path y false en caso contrario. La bandera watch habilita al cliente el fijar una vigilancia en el znodo.
- **getData(path, watch)**: retorna los datos y meta-datos asociados con el znode. ZooKeeper no habilita la bandera watch en caso el nodo no exista.

- **setData(path, data, version)**: escribe data[] en el znodo path si el número version es la versión actual del nodo.
- **getChildren(path, watch)**: retorna el conjunto de nombres de los hijos del znodo path.
- **sync(path)**: espera a que todas las operaciones pendientes al inicio de la operación se propaguen al servidor al que está conectado el cliente. El parámetro path es usualmente ignorado.

Todos los métodos cuentan con una versión sincrónica y asincrónica en la API. El cliente garantiza que los callbacks correspondientes a cada operación sean invocados en orden.

Debido a que todos los métodos requieren la ruta completa del znodo (path) sobre el cual operan, ZooKeeper no requiere complejidad adicional para acceder a los nodos.

Garantías de ZooKeeper

ZooKeeper tiene dos garantías básicas:

- **Escrituras Linearizables:** todas las solicitudes que actualicen el estado del ZooKeeper son serializables y respetan la precedencia.
- **Orden de Cliente FIFO:** todas las solicitudes de un cliente dado se ejecutan en el orden en que son enviados por el cliente.

A la definición de linearizable presentada le llamamos **A-linearizable** (linearizable asincrónica).

Debido a que sólo las actualizaciones son linearizables, ZooKeeper procesa operaciones de lectura localmente en cada réplica. Esto permite al servicio escalar linealmente a medida que se agregan servidores.

Ejemplos de Primitivas

ZooKeeper se puede usar para implementar algunas primitivas poderosas como las que mostraremos a continuación.

Gestión de la Configuración

ZooKeeper puede ser utilizado para implementar una configuración dinámica en aplicaciones distribuidas. En la forma más simple, la configuración está almacenada en un znodo *zc*. Los procesos comienzan con el pathname completo de *zc*. Los procesos obtienen su configuración leyendo *zc* y fijan la bandera de vigilancia como true para ser notificados en caso la configuración en *zc* sea actualizada.

Rendezvous

En los sistemas distribuidos no es siempre claro a priori cómo será la configuración final del sistema. Supongamos que un cliente quiere iniciar un proceso master y varios procesos worker, pero el iniciar procesos es hecho por un scheduler por lo que el cliente no sabe de antemano información que pueda dar a los procesos workers para conectarse al master. Manejaremos este escenario usando un znodo rendezvous *zr*, el cual es creado por el cliente. El cliente pasa el pathname completo de *zr* como un parámetro de inicio de los procesos. Cuando el master comienza, llena *zr* de información acerca de las direcciones y puertos que está usando. Cuando el worker inicia, lee *zr* con la vigilancia fijada en true. Si *zr* no ha sido llenado aún, espera a que sea notificado de la actualización de *zr*.

Membresía de Grupo

Aprovechamos el hecho de que los znodos efímeros nos permitan ver el estado de la sesión que lo creó para implementar las membresías de grupo. Designamos un nodo *zg* para que represente un grupo. Cuando un proceso miembro de dicho grupo inicia, crea un znodo efímero hijo bajo *zg*. Los procesos pueden poner su información en el hijo del znodo creado.

Si el proceso que creo el hijo de *zg* falla o termina, el nodo que creo debajo de zg es removido automáticamente.

Los procesos pueden obtener información del grupo simplemente listando los hijos de *zg.*

Bloqueos Simples

La implementación de bloqueo más simple utiliza “archivos de bloqueo”. El bloqueo es representado por un znodo. Para adquirir el bloqueo, el cliente intenta crear el znodo designado con la bandera EPHEMERAL. Si la creación es exitosa, el cliente mantiene el bloqueo. En caso contrario, el cliente puede leer el znodo con la bandera de vigilancia fijada para ser notificado si el líder actual muere. Un cliente libera el bloqueo cuando muere o elimina explícitamente el znodo. Los otros clientes que estén esperando por el bloqueo lo intentar adquirir una vez observen al znodo siendo eliminado.

Problemas:

- Efecto de Manada: si hay muchos clientes esperando adquirir el bloqueo, todos ellos se disputarán el bloqueo aunque sólo uno lo obtenga al final.
- Sólo implementa un bloqueo exclusivo.

Bloqueo Simple sin Efecto de Manada

Definimos un znodo *l* para implementar estos bloqueos. Intuitivamente se alinean todos los clientes que soliciten el bloqueo y cada uno lo obtiene acorde al orden de llegada de su solicitud. Los clientes que deseen el bloqueo harán lo siguiente:

Lock

```
1 n = create(l + "/lock-", EPHEMERAL|SEQUENTIAL)
2 C = getChildren(l, false)
3 if n is lowest znode in C, exit
4 p = znode in C ordered just before n
5 if exists(p, true) wait for watch event
6 goto 2
```

Unlock

```
1 delete(n)
```

El uso de la bandera SEQUENTIAL ordena los intentos de los clientes de adquirir el bloqueo.

Al usar la bandera EPHEMERAL en la creación, los procesos que fallen limpiarán automáticamente toda solicitud de obtener el bloqueo o liberarán cualquier bloqueo que tengan.

En resumen, este esquema de bloqueo posee las siguientes ventajas:

1. Remover un znodo sólo causa que un cliente se despierte para obtener el bloqueo por lo que no obtenemos el efecto de manada.
2. No hay votaciones de tiempos muertos.
3. Debido a la manera en que se implementó el bloqueo, podemos ver al buscar en los datos del ZooKeeper la cantidad de contención de bloqueo, bloqueos de roturas y debugear los problemas de bloqueo.

Bloqueos de Lectura/Escritura

Write Lock

```
1  n = create(l + "/write-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if n is lowest znode in C, exit
4  p = znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 2
```

Read Lock

```
1  n = create(l + "/read-", EPHEMERAL|SEQUENTIAL)
2  C = getChildren(l, false)
3  if no write znodes lower than n in C, exit
4  p = write znode in C ordered just before n
5  if exists(p, true) wait for event
6  goto 3
```

El proceso de desbloqueo es el mismo que en el caso global.

Debido a que los bloqueos de lectura pueden ser compartidos, las líneas 3 y 4 varía debido a que sólo znodos de bloqueo de escritura anteriores previenen que el cliente obtenga un bloqueo de lectura.

Barreras Dobles

Las barreras dobles permiten a los clientes sincronizar el comienzo y fin de una computación. Cuando una cantidad de procesos suficiente se hayan unido a la barrera, definida por un límite en la barrera, los procesos inician su computación y dejan la barrera cuando hayan terminado. Representamos la barrera con un znodo *b*. Todo proceso se registra con *b* creando un znodo hijo de *b* al entrar y eliminan dicho nodo al salir. Los procesos pueden seguir entrando a la barrera aún cuando la cantidad de hijos de *b* excede el límite de la barrera. Los procesos pueden dejar la barrera cuando todos los procesos hayan removido sus hijos. Se usan vigilancias para esperar eficientemente que se cumplan las condiciones de entrada y salida. Para la entrada, los procesos vigilan la existencia de un hijo *ready* de b que será creado por el proceso que ocasione que el número de hijos exceda el límite. Para la salida, los procesos vigilan que un hijo en particular desaparezca.

Aplicaciones de ZooKeeper

El Servicio de Búsqueda (Fetching)

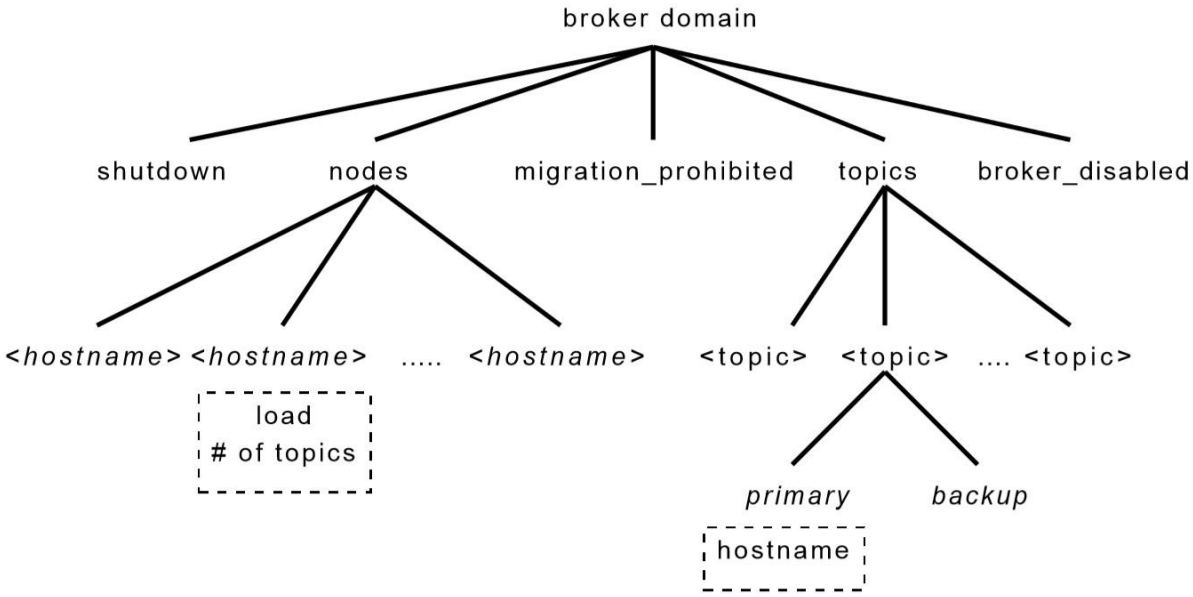
- El servicio de búsqueda (FS) es parte del rastreador de Yahoo! y está actualmente en producción.
- Posee procesos maestro que comandan los procesos de búsqueda de página. El maestro provee a los buscadores la configuración y ellos responden informando de su estado y salud.
- Las principales ventajas de usar ZooKeeper son la recuperación luego de fallos del maestro, garantizar la disponibilidad a pesar de errores y disociar el cliente del maestro, permitiéndoles dirigir sus solicitudes a servidores sanos simplemente leyendo su estado del ZooKeeper.
- FS usa ZooKeeper principalmente para la **configuración de los metadatos** y la elección del maestro (**elección de un líder**).

Katta

- Katta es un indexador distribuido que utiliza ZooKeeper para la coordinación.
- Katta divide el trabajo de indexar en fragmentos. Un servidor maestro asigna fragmentos a los esclavos y sigue el progreso.
- Como los esclavos pueden fallar, por lo que el maestro debe redistribuir la carga a medida que entran o salen servidores.
- El maestro también puede fallar, por lo que otros servidores deben estar listos para asumir el cargo.
- Katta usa ZooKeeper para seguir el estado del maestro y los esclavos (**membresía de grupo**) y manejar la falla del maestro (**elección de líder**). Tambien para rastrear y propagar las asignaciones de fragmentos a los esclavos (**gestión de configuración**).

Agente de Mensajes de Yahoo! (YMB)

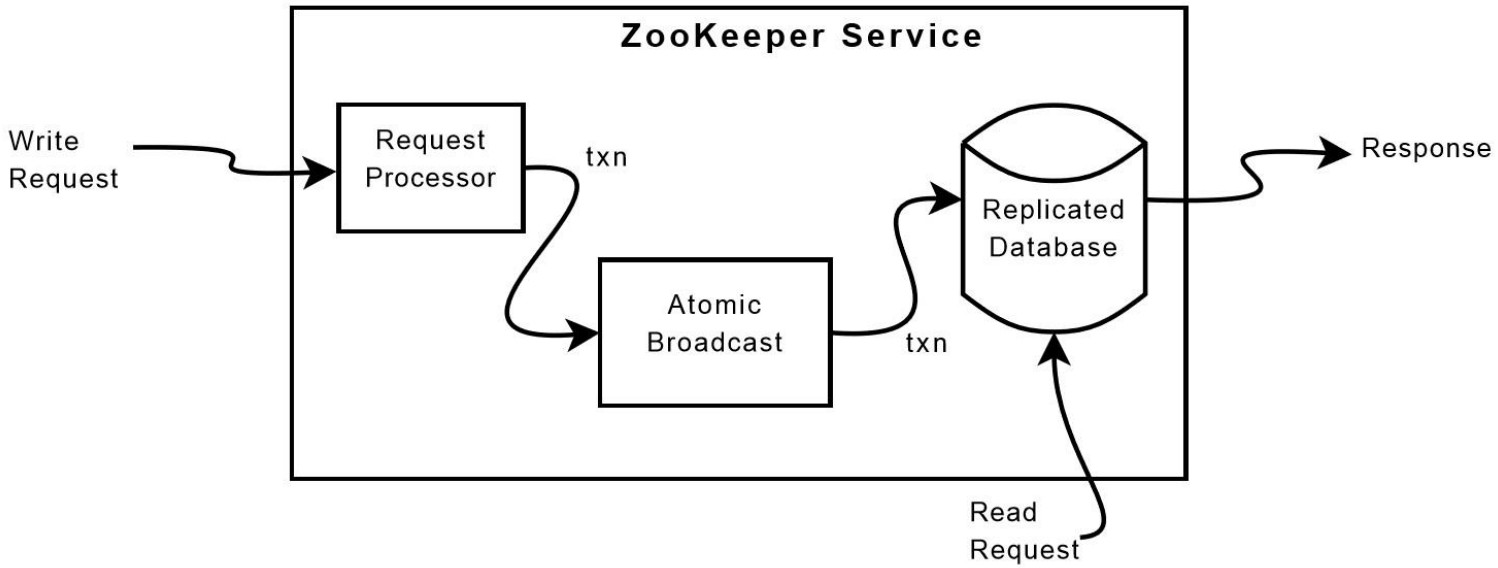
- El sistema maneja miles de temas donde los usuarios pueden publicar o recibir mensajes.
- Los temas deben ser distribuidos entre un conjunto de servidores para proveer la escalabilidad.
- YMB utiliza ZooKeeper para manejar la distribución de temas **(metadatos de configuración)**, lidiar con fallas de las máquinas **(detección de fallas** y **membresías de grupo)**, y controlar las operaciones de sistema.



- Cada servidor YMB crea un znodo efímero debajo de *nodes* con la información de su carga y estado proporcionando tanto la información de membresía de grupo como la de su estado a través de ZooKeeper.
- Los nodos *shutdown* y *migration_prohibited* son monitoreados por todos los servidores que componen el servicio, y permiten un control centralizado del YMB.
- El directorio *topics* tiene un znodo hijo por cada tema manejado por YMB.
- Los hijos de los znodos de cada tema, *primary* y *backup*, no sólo permiten a los servidores descubrir los servidores que se encargan de cierto tema, también manejan la elección del líder y fallos del servidor.

Implementación de ZooKeeper

- ZooKeeper provee una alta disponibilidad replicando los datos del ZooKeeper en cada servidor del servicio.



- Cuando se recibe una solicitud el servidor la prepara para su ejecución (request processor).

- Si la solicitud requiere coordinación entre los servidores, entonces ellos usan un protocolo de acuerdo (atomic broadcast), y finalmente comenten los cambios a la base de datos del ZooKeeper completamente replicada en todos los servidores del conjunto.
- En caso de solicitudes de lecturas, simplemente se lee la base de datos local.
- Cada znodo del árbol de datos almacena a lo mucho 1 MB de datos por defecto, pero este valor se puede modificar.
- Los clientes se conectan con exactamente un servidor par emitir sus solicitudes.
- Como parte del protocolo de acuerdo, las solicitudes de escritura son dirigidas a un solo servidor (el líder), quien propone a los otros servidores (los seguidores) los cambios en el estado.

Implementación: Procesador de Solicitudes

- Debido a que la capa de mensajes es atómica, garantizamos que las replicas locales nunca diverjan. Aunque algunos servidores podrían haber aplicado más transacciones que otros en algún punto en el tiempo.
- Las transacciones son *idempotentes*.
- Cuando el líder recibe una solicitud de escritura, calcula cuál será el nuevo estado del sistema al aplicar la escritura y lo transforma en una transacción que captura el nuevo estado.
- El nuevo estado debe ser calculado pues podrían haber transacciones pendientes que aún no han sido aplicadas a la base de datos.

Implementación: Transmisión (Broadcast) Atómica

- El líder ejecuta las solicitudes y transmite los cambios al estado del ZooKeeper a través de **Zab**, un protocolo de transmisión atómica.
- Zab usa por defecto quórums de mayoría para decidir acerca de una propuesta, por lo que el ZooKeeper sólo puede trabajar si la mayoría de servidores están correctos.
- Debido a que los cambios en el estado depende de la aplicación de cambios previos, Zab provee una garantía de orden más fuerte que las transmisiones atómicas regulares.
- Se utiliza TCP para el transporte por lo que el orden de los mensajes es mantenido por la red.
- Se usa un registro para mantener el seguimiento de las propuestas como un registro write-ahead para la base de datos en la memoria de tal manera que no se escriban los mensajes dos veces en el disco.

Implementación: Base de Datos Replicada

- ZooKeeper utiliza snapshots periódicos para evitar que el tiempo de recuperación de un servidor luego de una falla no sea demasiado largo.
- Los snapshots de ZooKeeper son llamados *fuzzy snapshots* pues no bloquean el estado del ZooKeeper para tomar el snapshot.
- El snapshot de ZooKeeper escanea el árbol leyendo automáticamente cada dato y metadato en los znodos y los escribe en el disco.
- Debido a que el fuzzy snapshot resultante pudo haber aplicado algunos de los cambios de estado durante la generación del snapshot, el resultado podría no corresponder al estado del ZooKeeper original. Sin embargo, gracias a la idempotencia de los cambios de estado, esto no causa problemas mientras los cambios se apliquen en orden.

Implementación: Interacción Servidor-Cliente

- Los servidores manejan las notificaciones correspondientes a las vigilancias de manera local. Sólo el servidor al que está conectado cierto cliente rastreará y provocará las notificaciones para dicho cliente.
- Las solicitudes de lectura se manejan localmente en cada servidor para obtener un excelente rendimiento de lectura.
- Cada solicitud de lectura es procesada y marcada con un *zxid* que corresponde a la última transacción vista por el servidor. Esto es para definir un orden parcial de solicitudes de lectura respecto a las solicitudes de escritura.
- Un inconveniente de usar lecturas rápidas es el no garantizar un orden de precedencia para las lecturas, es decir, una operación de lectura podría retornar un valor pasado.

- No todas las aplicaciones necesitan un orden de precedencia, pero para aquellas que lo necesiten se ha implementado la primitiva sync.
- El cliente llama a sync seguido de la operación de lectura para asegurarse de conseguir el último valor actualizado.
- Los servidores procesan las solicitudes de los clientes en orden FIFO.
- Las respuestas incluyen el zxid relativo a la respuesta. Incluso los heartbeats durante intervalos de inactividad incluyen el último zxid visto por el servidor conectado al cliente.
- Si el cliente se conecta a un nuevo servidor, este servidor debe asegurarse que su vista de los datos del ZooKeeper sea al menos tan reciente como el del cliente revisando el último zxid del cliente con su propio zxid.
- El líder determina que ha habido un error en la sesión de un cliente si ningún servidor recibe algo de la sesión del cliente dentro del timeout de la sesión.

Conclusiones

- ZooKeeper toma un enfoque wait-free para el problema de la coordinar procesos en sistemas distribuidos al exponer objetos wait-free a los clientes.
- ZooKeeper posee varias aplicaciones dentro y fuera de Yahoo!
- ZooKeeper obtiene un gran rendimiento en cargas de trabajo en las que dominen las operaciones de lectura.
- La propiedad wait-free es esencial para el alto rendimiento.