Zookeeper

Координатор распределенных систем

Зачем нужен Zookeeper

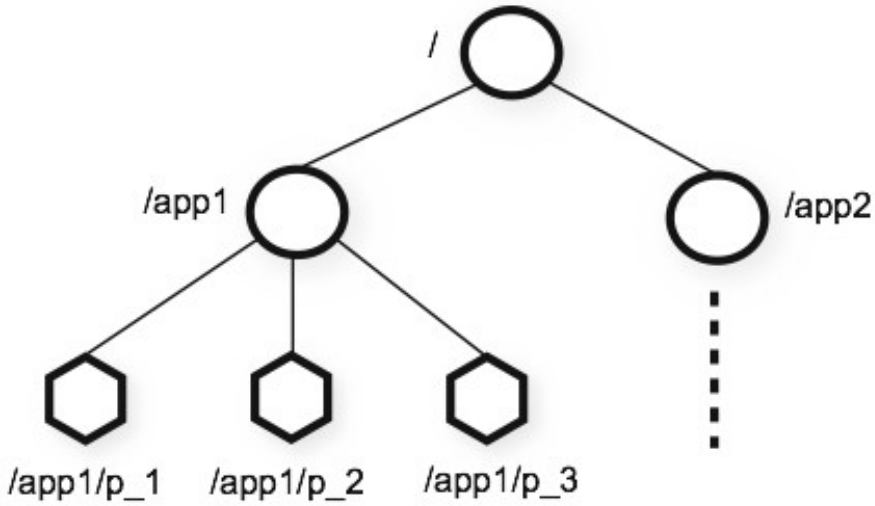
- Синхронизация узлов распределенной системы
- Хранилище конфигурации
- Гарантия порядка сообщений
- Набор удобных примитивов для реализации своих алгоритмов
- Надежность
- Скорость

Total order, casual oder

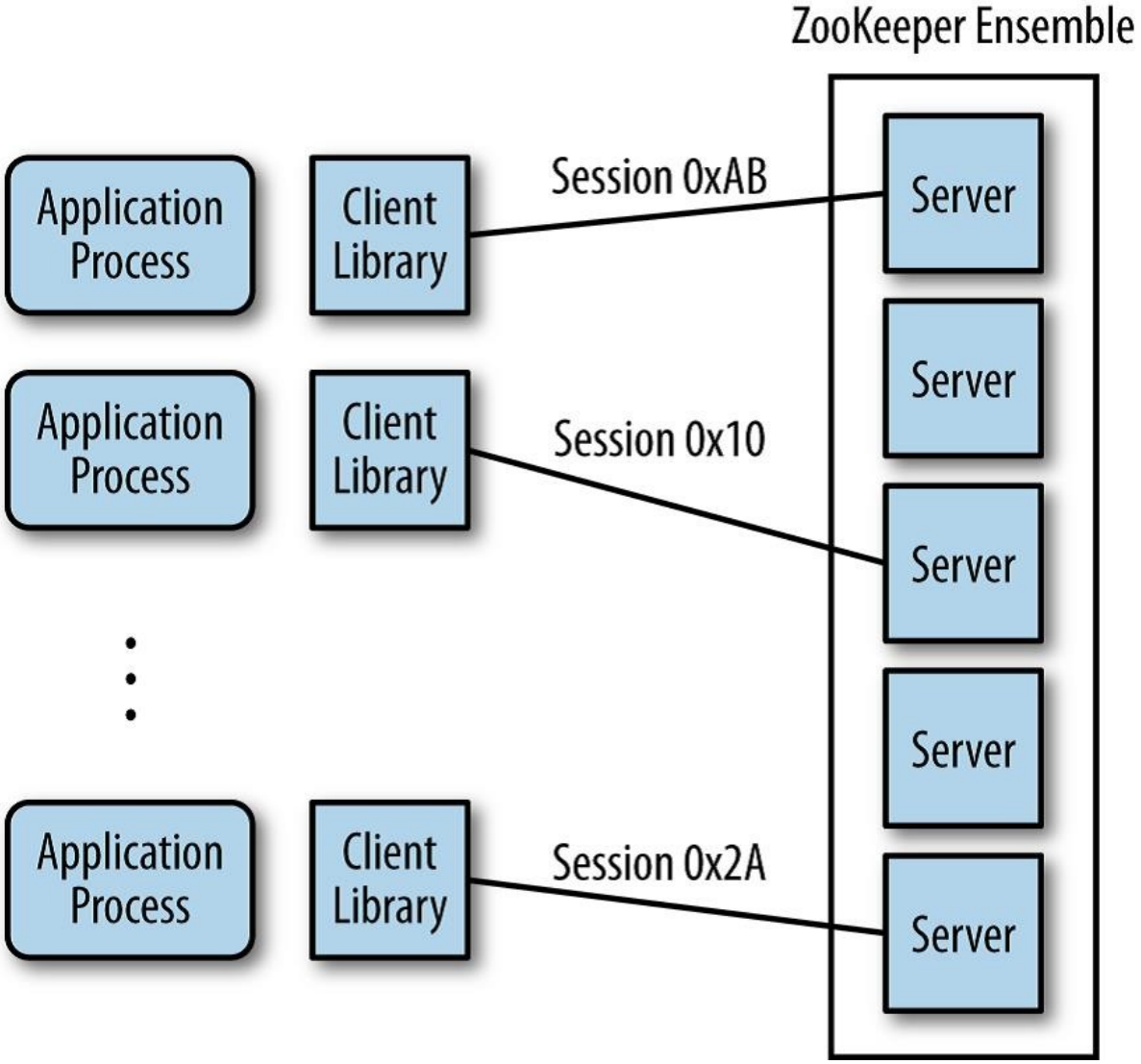
- Total order гарантирует следующее :
  - Если сообщение А доставлено раньше Б одним сервером, то оно будет доставлено раньше Б всеми серверами.
  - Если А и Б – доставленые сообщения, то либо А доставлено раньше Б, либо наоборот
- Casual order дает следующую гарантию :
  - Если отправитель отправил сообщение А раньше чем Б, то и доставлено оно будет раньше чем Б.

Модель данных

- Модель данных – дерево
- Узел – Znode :
  - Persistent
  - Ephemeral
  - Sequence Node
- Операции чтения/записи атомарны
- Каждый узел имеет содержимое (массив байт) по умолчанию максимальный размер 1MB



Архитектура



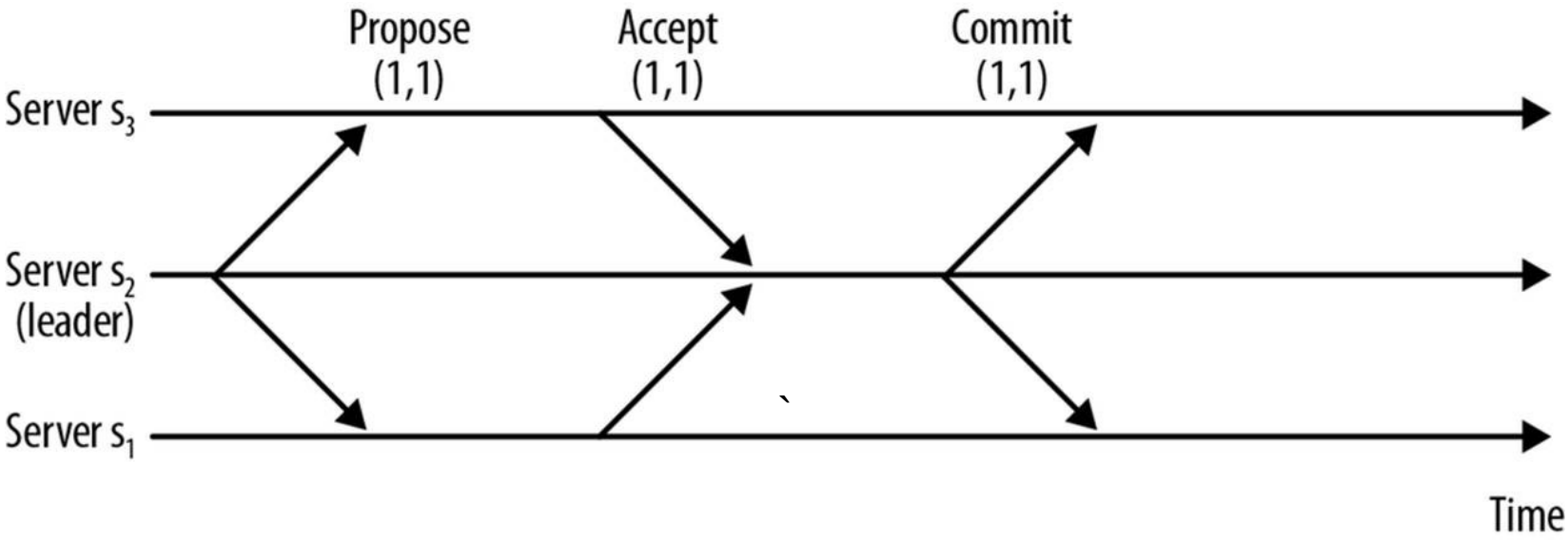
Архитектура

- Кластер Zookeeper состоит из нечетного числа узлов
- Каждый узел хранит в памяти полный слепок данных
- Для отказоустойчивости все операции записываются в transaction log
- Все узлы одинаковы, но один из узлов является “Лидером”

Основные идеи

- Команды на чтение выполняются локально
- Для синхронизации состояния сервера используется команда sync
- Все команды модифицирующие данные передаются лидеру
- Лидер обрабатывает команды последовательно
- Каждая команда рассылается всем узлам, после получения кворума считается выполненной

Цикл обработки команд



1. Leader sends propose messages
2. Followers ack the proposal
3. Leader commits the proposal

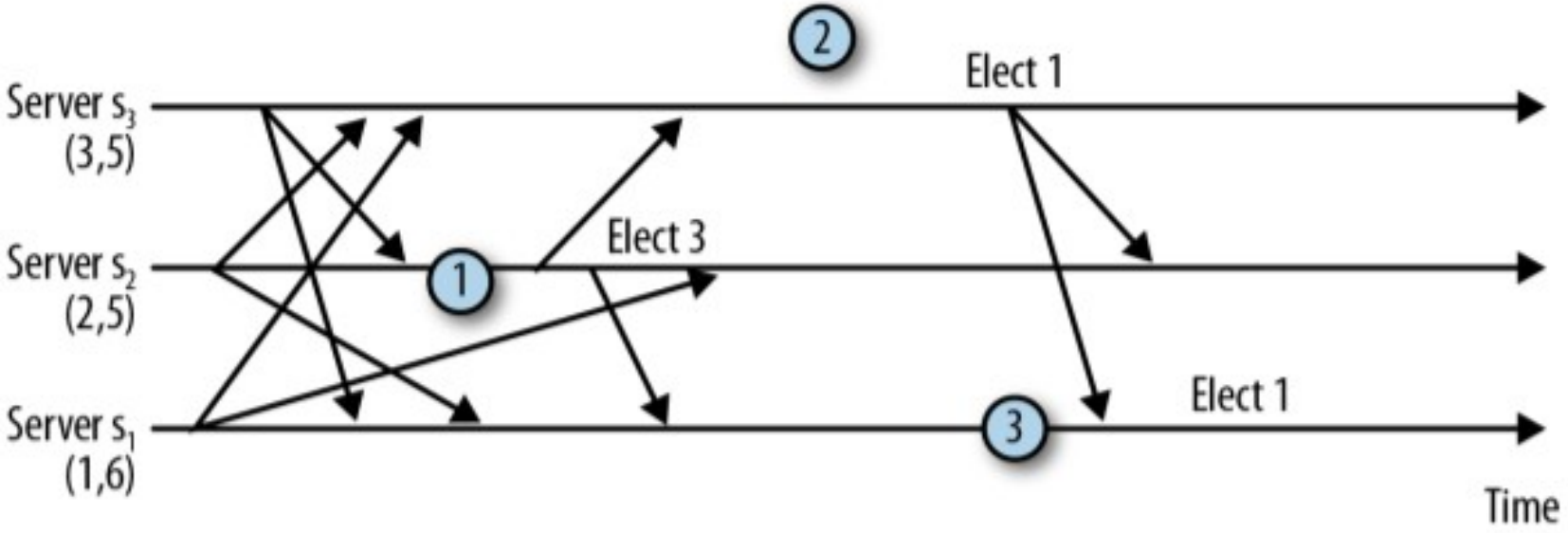
Транзакция

- Каждая операция изменяющая состояние передается лидеру и он создает транзакцию
- Транзакции присваивается номер Zxid – увеличивается на 1 текущий counter который хранится у лидера
- Zxid состоит из двух частей (каждая по 32 бита) :
  - Epoch (номер эпохи)
  - Counter (номер транзакции в эпохе)
- Транзакции идемпотентны (могут быть применены несколько раз к состоянию)

Старт кластера – leader election

- Каждый сервер отправляет всем серверам свой голос – номер сервера и номер на этом сервере последней транзакции
- Если сервер получает сообщение в котором видит что пришел номер больший номер транзакции, то он переголосует
- Когда сервер получает кворум голосов, он считается выбранным
- В качестве таймаута ожидания используется по умолчанию 200 мс

Выборы лидера



1. Server s_2 receives vote (3,5) and changes its vote, forming a quorum. It elects server s_3.
2. Server s_3 receives vote (1,6), but it takes some time to send a new batch of notifications.
3. Server s_1 elects itself leader once it receives the vote of server s_3.

Синхронизация, активация лидера

- После выборов нового лидера :
- Каждый сервер follower подключается к лидеру и получает недостающие транзакции, либо полный снапшот всего массива данных
- Лидер посылает всем команду NEW_LEADER
- Лидер дожидается кворума (получет от всех ACK)
- Лидер высылает COMMIT

Синхронизация

```
1  Leader L:
2  L.lastZxid ← ⟨L.lastZxid.epoch + 1, 0⟩
3  upon receiving FOLLOWERINFO(f.lastZxid) message from a follower f do
4      Send NEWLEADER(L.lastZxid) to f
5      if f.lastZxid ⪯ L.history.lastCommittedZxid then
6          if f.lastZxid ≺ L.history.oldThreshold then
7              Send a SNAP message with a snapshot of the whole database of L
8          else
9              Send a DIFF({committed transaction ⟨v, z⟩ ∈ L.history : f.lastZxid ≺ z})
10         end
11     else
12         Send a TRUNC(L.history.lastCommittedZxid) message to f
13     end
14 end
15 upon receiving ACKNEWLEADER messages from some quorum of followers do
16     goto Phase 3     // Algorithm 3
17 end
```

Синхронизация

```
18 Follower F:
19 Connect to its prospective leader L
20 Send the message FOLLOWERINFO(F.lastZxid) to L
21 upon L denies connection do
22     F.state ← election and goto Phase 0
23 end
24 upon receiving NEWLEADER(newLeaderZxid) from L do
25     if newLeaderZxid.epoch < F.lastZxid.epoch then
26         F.state ← election and goto Phase 0
27     end
28     upon receiving a SNAP, DIFF, or TRUNC message do
29         if got TRUNC(lastCommittedZxid) then
30             Abort all proposals from lastCommittedZxid to F.lastZxid
31         else if got DIFF(H) then
32             Accept all proposals in H, in order of zxids, then commit all
33         else if got SNAP then
34             Copy the snapshot received to the database, and commit the changes
35         end
36         Send ACKNEWLEADER
37         goto Phase 3     // Algorithm 3
38     end
39 end
```

Обработка запросов

1 *Leader L*:
2 **upon** receiving a write request v **do**
3 Propose $\langle e', \langle v, z\rangle\rangle$ to all followers in Q, where $z = \langle e', c\rangle$, such that z succeeds all zxid values previously broadcast in e' (c is the previous zxid's counter plus an increment of one)
4 **end**
5 **upon** receiving ACK($\langle e', \langle v, z\rangle\rangle$) from a quorum of followers **do**
6 Send COMMIT($e', \langle v, z\rangle$) to all followers
7 **end**
8 *// Reaction to an incoming new follower:*
9 **upon** receiving FOLLOWERINFO(e) from some follower f **do**
10 Send NEWEPOCH(e') to f
11 Send NEWLEADER($e', L.$`history`) to f
12 **end**
13 **upon** receiving ACKNEWLEADER from follower f **do**
14 Send a COMMIT message to f
15 $Q \leftarrow Q \cup \{f\}$
16 **end**

Обработка запросов

17 *Follower F*:
18 **if** F is leading **then** Invokes *ready*(e')
19 **upon** receiving proposal $\langle e', \langle v, z\rangle\rangle$ from L **do**
20 Append proposal $\langle e', \langle v, z\rangle\rangle$ to F.`history`
21 Send ACK$(\langle e', \langle v, z\rangle\rangle)$ to L
22 **end**
23 **upon** receiving COMMIT$(e', \langle v, z\rangle)$ from L **do**
24 **while** there is some outstanding transaction $\langle v', z'\rangle \in F$.`history` such that $z' \prec_z z$ **do**
25 Do nothing (wait)
26 **end**
27 Commit (deliver) transaction $\langle v, z\rangle$
28 **end**

Сессия

- Для коректной работы ephemeral узлов и отслеживания изменений в zookeeper-е центральный сервер отслеживет всех подключенных клиентов
- Каждый клиент поддерживает сессию – отправляет heartbeat
- Лидер опрашивает все сервера в два раза чаще – каждый сервер высылает ему список подключенных в данный момент клиентов

Хранение данных

- Вся модель хранится в памяти
- Каждая транзакция сначала записывается во write ahead log а потом применяется к структуре в памяти
- Регулярно сохраняется полный snapshot модели на диск

Watch (подписка на изменения)

- Клиент может подписаться на изменения происходящие с узлом
- Подписка срабатывает 1 раз и требует обновления после каждой нотификации
- Установить подписку можно одновременно с получением данных об узле, данная операция является атомарной
- В случае если сессия пропадает – все подписки исчезают

API, подключение

- Создаем экземпляр класса Zookeeper, например

```
ZooKeeper(String connectString,
          int sessionTimeout,
          Watcher watcher
         )
```

- ConnectString – список серверов с портами через запятую
- SessionTimeout – timeout сессии в МС
- Watcher – callback для обработки событий относящихся к сессии

Получение данных об узлах

- getChildren – получить список дочерних узлов
- getData – получить информацию о содержимом узла
- exists – получить информацию об узле если он есть, или null
- sync – принудительно обновить данные на данном сервере zookeeper

Изменение данных

- create – создать узел
- delete – удалить узел
- setData – изменить данные на узле, версия передается для контроля, было ли изменение которое мы пропустили.
- multi – выполнить несколько операций или ни одну из ни
- transaction – обертка вокруг метода multi

Пример

```
ZooKeeper zoo = new ZooKeeper("127.0.0.1:2181", 3000, this);
zoo.create("/servers/s",
  “data”.getBytes(),
  ZooDefs.Ids.OPEN_ACL_UNSAFE ,
  CreateMode.EPHEMERAL_SEQUENTIAL
);
List<String> servers = zoo.getChildren("/servers",  this);
for (String s : servers) {
    byte[] data = zoo.getData("/servers/" + s, false, null);
    System.out.println("server " + s + " data=" + new String(data));
}
```