分布式面向列的数据库HBase

Northeastern University
HORIZON 昊宸科技

目录

目标

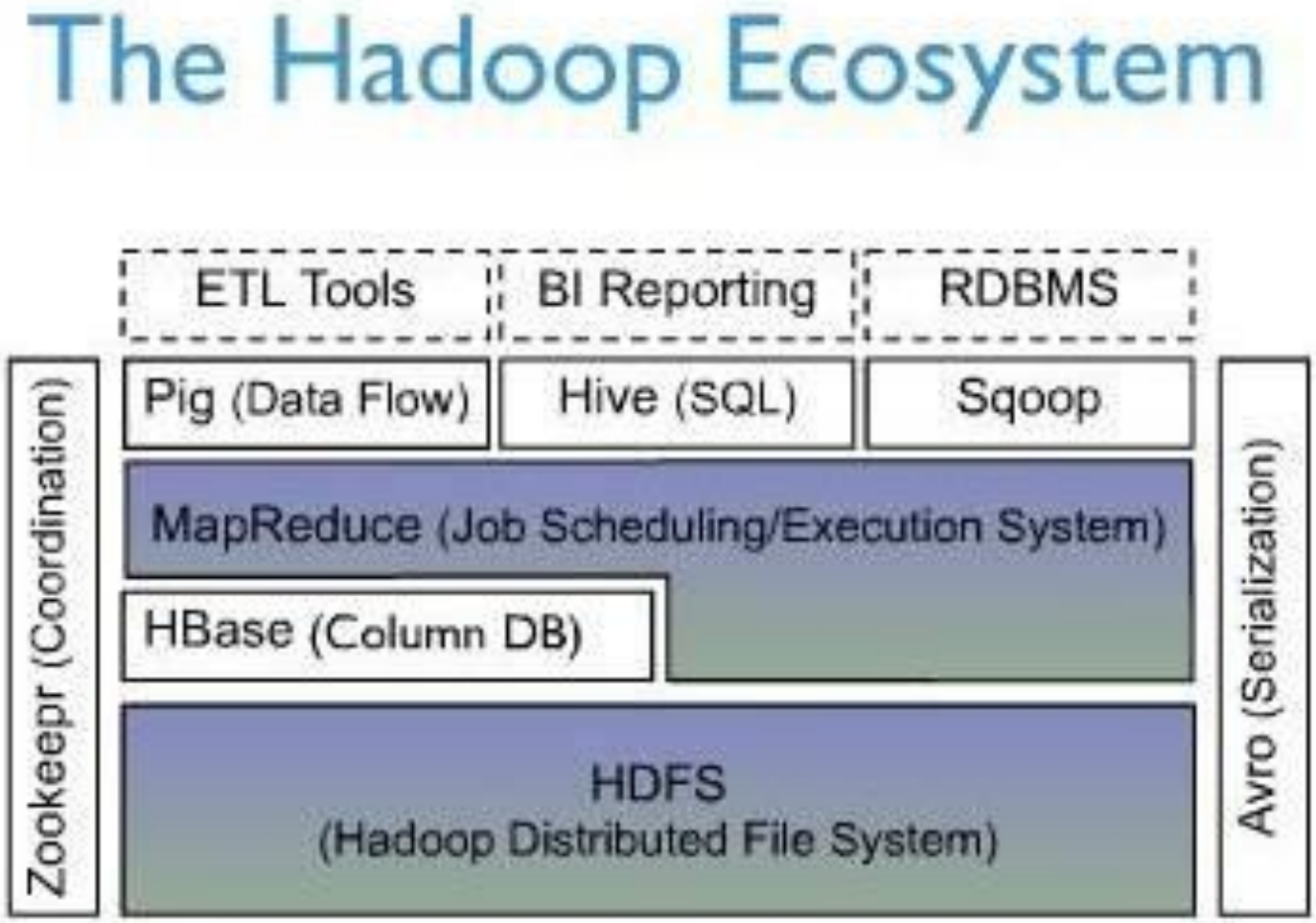
- 搭建HBase分布式环境。
- 了解HBase的逻辑模型。
- 了解HBase的物理模型。
- 简单的使用HBase shell。

HBase概述

- HBase是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用HBase技术可在廉价PC Server上搭建起大规模结构化存储集群。
- HBase是Google Bigtable的开源实现，类似Google Bigtable利用GFS作为其文件存储系统，HBase利用Hadoop HDFS作为其文件存储系统；Google运行MapReduce来处理Bigtable中的海量数据，HBase同样利用Hadoop MapReduce来处理HBase中的海量数据；Google Bigtable利用 Chubby作为协同服务，HBase利用Zookeeper作为对应。
- HDFS只适合一次存入，多次读取的场合。不支持数据的随机读写操作。HBase的出现解决了HDFS的这些痛点。

Hbase概述

- HBase位于结构化存储层，
- Hadoop HDFS为HBase提供了高可靠性的底层存储支持，
- Hadoop MapReduce为HBase提供了高性能的计算能力，
- Zookeeper为HBase提供了稳定服务和failover机制。
- Pig和Hive还为HBase提供了高层语言支持，使得在HBase上进行数据统计处理变的非常简单。
- Sqoop则为HBase提供了方便的RDBMS数据导入功能，使得传统数据库数据向HBase中迁移变的非常方便。

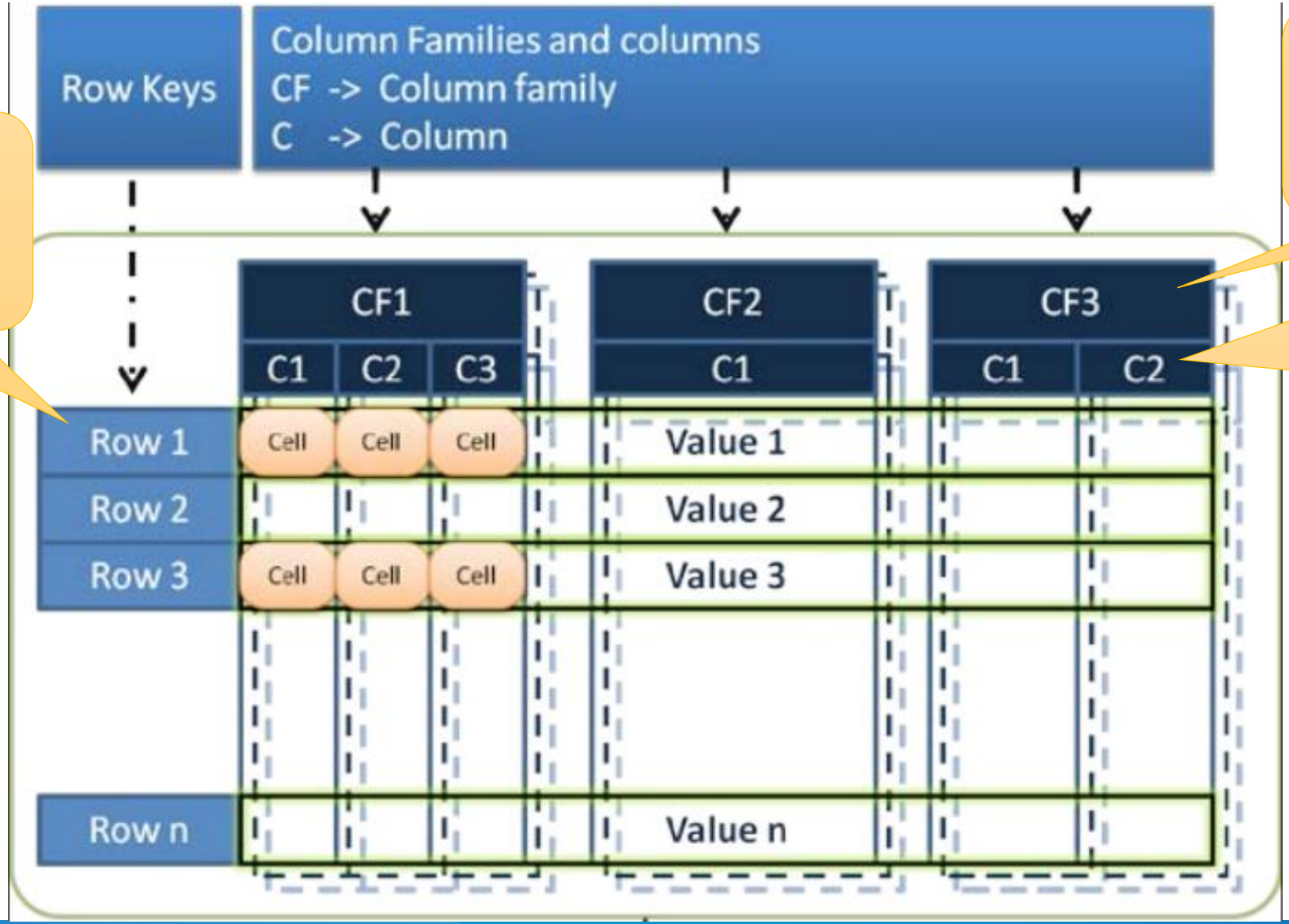


Hbase概述-HBase vs RDBMS

- 数据类型
  - ✓ HBase只有字符串（字节数组）
  - ✓ RDBMS有丰富的数据类型。
- 数据操作：
  - ✓ HBase只支持增删改查
  - ✓ RDBMS支持SQL语句。
- 存储模式
  - ✓ HBase基于列存储。
  - ✓ RDBMS基于行存储。
- 数据更新
  - ✓ HBase数据有多个版本。
  - ✓ RDBMS更新后覆盖。
- 扩展性
  - ✓ HBase具有很高的扩展性。RDBMS扩展性有限。

HBase数据模型-逻辑视图

每一行有一个Rowkey。是表的主键，rowkey的设计非常重要。



Column Family:列族，包含一个或多个列。

Column:列，属于某个列族。

HBase数据模型-基本概念

➢rowkey

- ✓ row key是用来检索记录的主键。访问hbase table中的行，只有三种方式：通过单个row key访问；通过row key的range；全表扫描。
- ✓ row key行键 (Row key)可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在hbase内部，row key保存为字节数组。
- ✓ 存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分利用排序存储这个特性，将经常一起读取的行存储放到一起。

➢列族(column family)

- ✓ hbase表中的每个列，都归属与某个列族。
- ✓ 列族是表的schema的一部分(而列不是)，必须在使用表之前定义。
- ✓ 列名都以列族作为前缀。

HBase数据模型-基本概念

➢单元(cell)

- ✓ HBase中通过row和columns确定的为一个存贮单元称为cell。
- ✓ cell中的数据是没有类型的，全部是字节码形式存贮。

➢时间戳(timestamp)

- ✓ 每个cell都保存着同一份数据的多个版本。
- ✓ 版本通过时间戳来索引。
- ✓ 时间戳可以由hbase(在数据写入时自动)赋值，此时是精确到毫秒的当前系统时间。也可以由客户显式赋值。
- ✓ 每个cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。
- ✓ hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。

HBase数据模型-物理视图

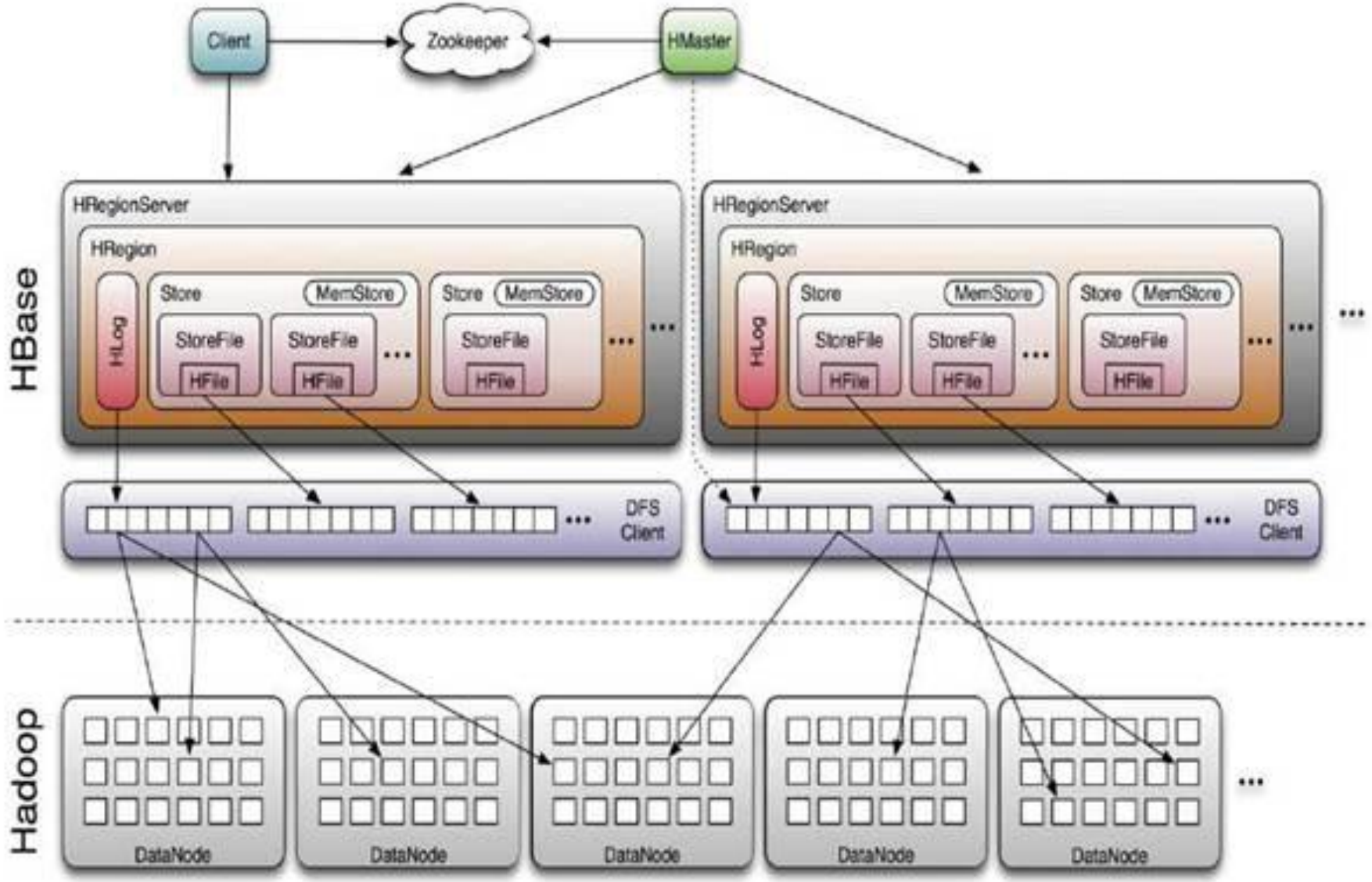
	Column Family :: CF1		Column Family :: CF2	
Row Keys	Col-1	Col-2	Col-1	Col-2
ROW-1	David	982 765 2345		
ROW-2			John	
ROW-3	Elan			909 451 4587 863 441 4123
ROW-4	Maria	763 451 4587 863 341 4123		

Physical Representation

Physical Representation

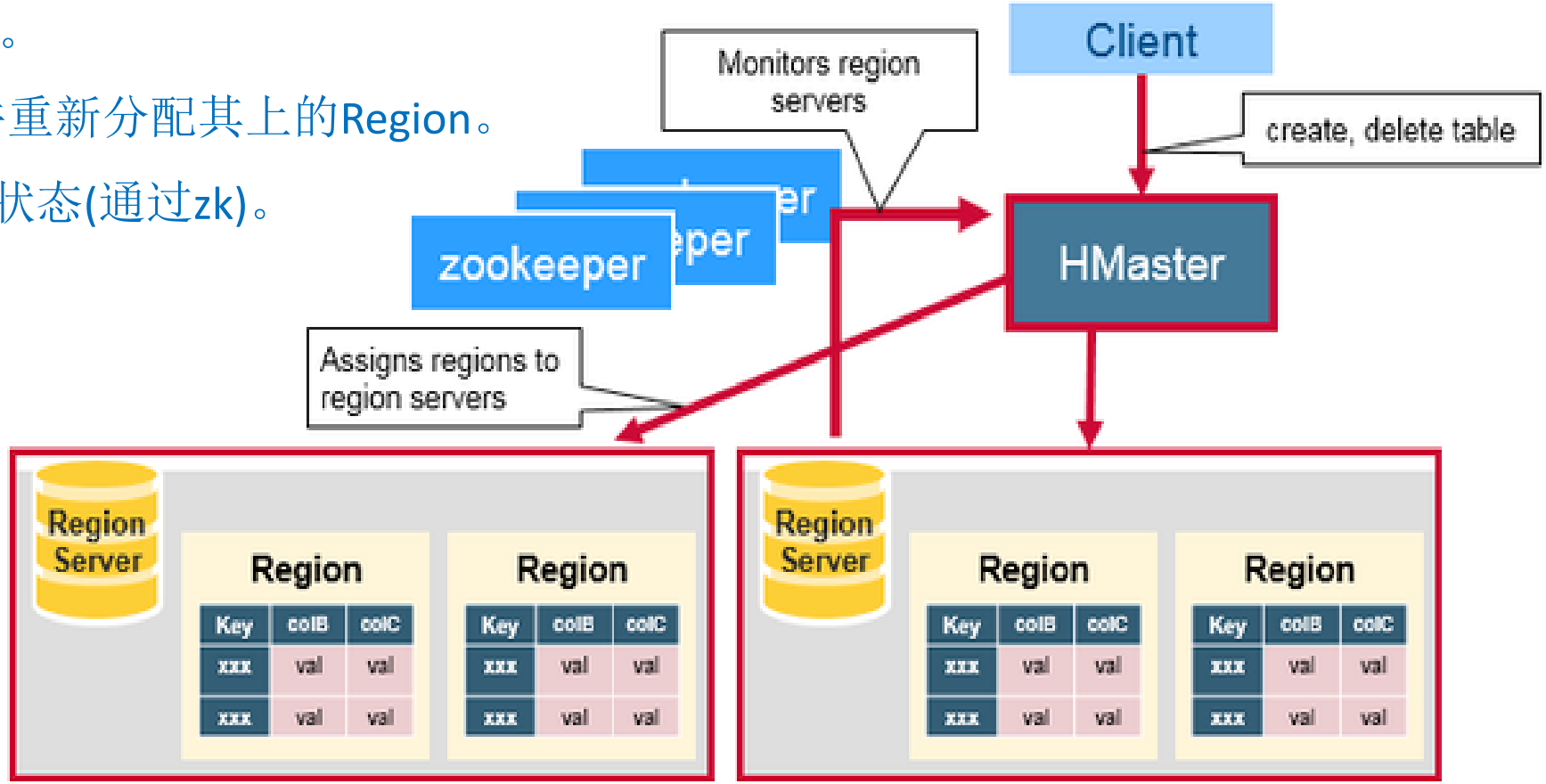
ROW-1 : CF1 : Col1 : TS1 : David	ROW-2 : CF2 : Col1 : TS1 : John
ROW-1 : CF1 : Col2 : TS1 : 982 765 2345	ROW-3 : CF2 : Col2 : TS1 : 909 451 4587
ROW-3 : CF1 : Col1 : TS1 : Elan	ROW-3 : CF2 : Col2 : TS2 : 823 441 4123
ROW-4 : CF1 : Col1 : TS1 : Maria	
ROW-4 : CF1 : Col2 : TS1 : 763 451 4587	
ROW-4 : CF1 : Col2 : TS2 : 863 341 4123	

HBase架构


Client
Zookeeper
HMaster
HBase
HRegionServer
HRegion
HLog
Store
MemStore
StoreFile
HFile
DFS Client
Hadoop
DataNode

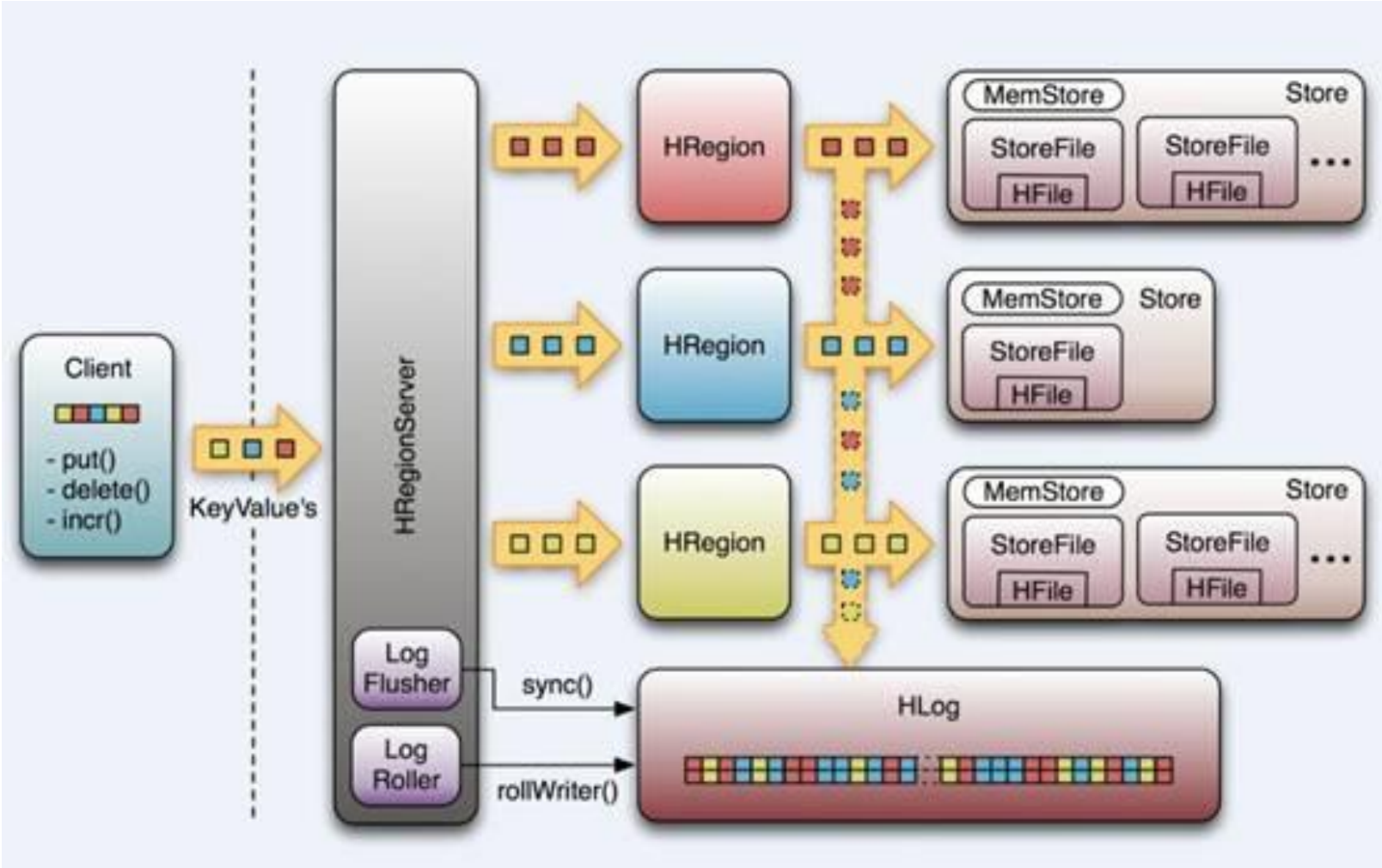
HBase架构-HMaster

- 管理用户的增删改查等操作。
- 协调RegionServer
  - ✓ 为RegionServer分配Region。
  - ✓ 发现失效的RegionServer并重新分配其上的Region。
  - ✓ 监控集群上所有RS的健康状态(通过zk)。



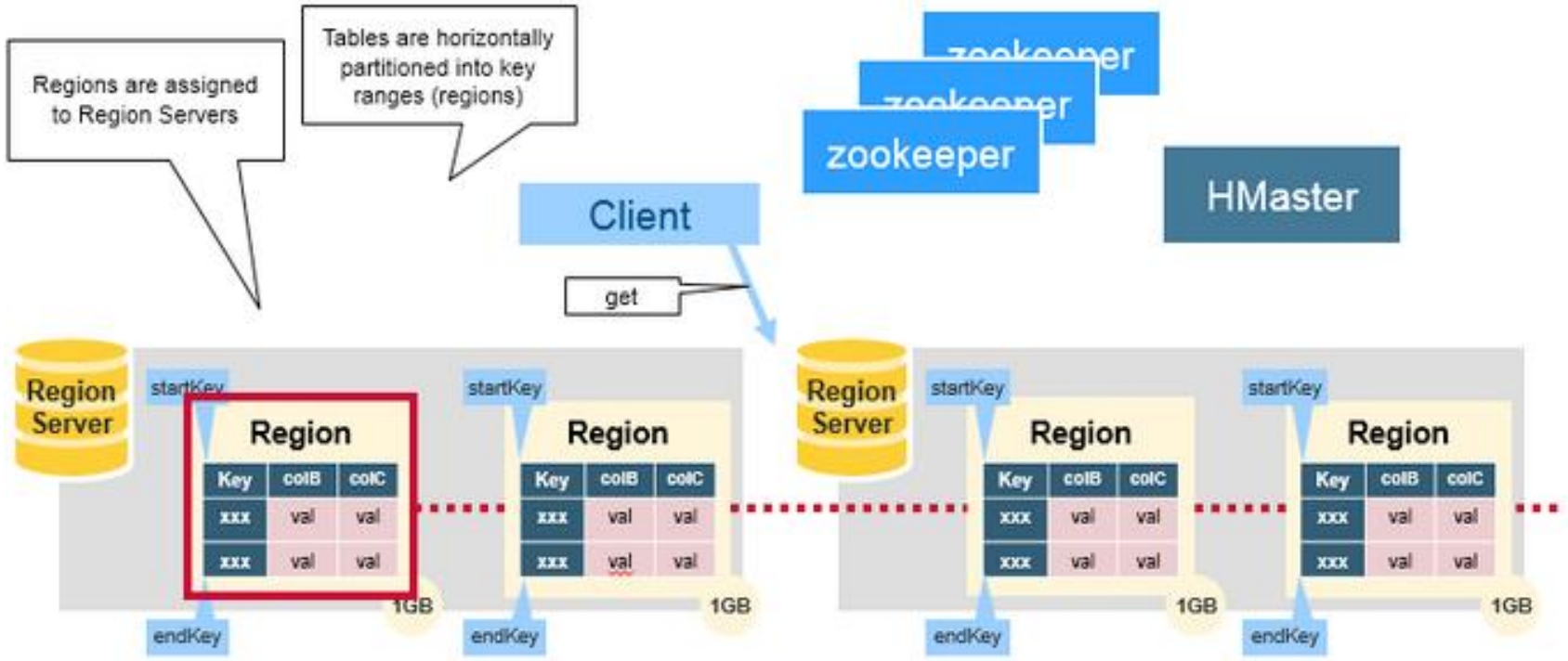
HBase架构-RegionServer

- ✓ 响应用户I/O请求,向HDFS文件系统中读写数据 。
- ✓ 内部管理了一系列HRegion对象。
- ✓ 每个HRegion对应了Table中的一个Region，HRegion中由多个HStore组成。
- ✓ 每个HStore对应了Table中的一个Column Family的存储。
- ✓ Hfile: HBase中KeyValue数据的存储格式，HFile是Hadoop的二进制格式文件。
- ✓ HRegionServer中都包含一个WAL(write a head log)。用于保存还未持久化存储的数据，用户数据的还原。
- ✓ RS用于切分过大的Region。



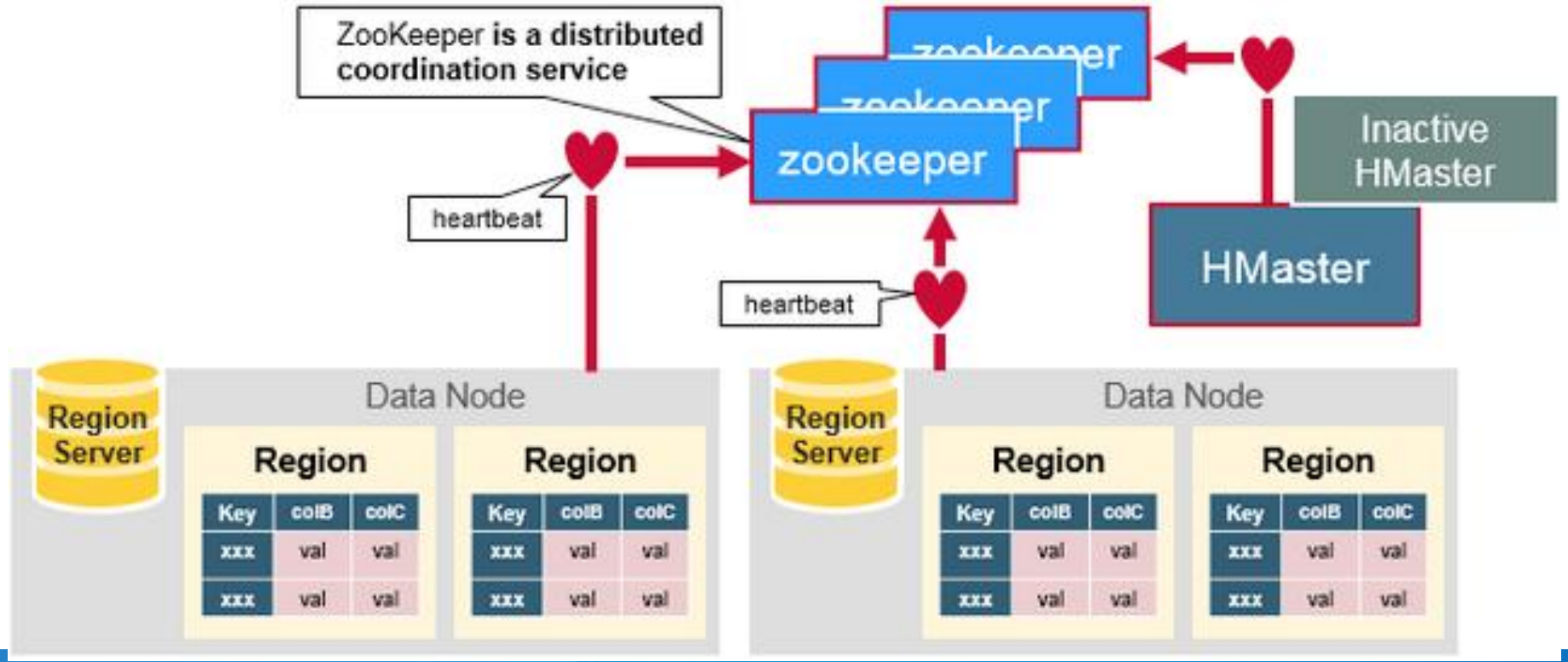
HBase架构-Region

- HBase使用RowKey将表水平切割成多个HRegion。
- 从HMaster的角度，每个HRegion都纪录了它的StartKey和EndKey。由于RowKey是排序的，因而Client可以通过HMaster快速的定位每个RowKey在哪个HRegion中。
- HRegion由HMaster分配到相应的HRegionServer中，然后由HRegionServer负责HRegion的启动和管理，和Client的通信，负责数据的读(使用HDFS)
- 每个HRegionServer可以同时管理1000个左右的HRegion。



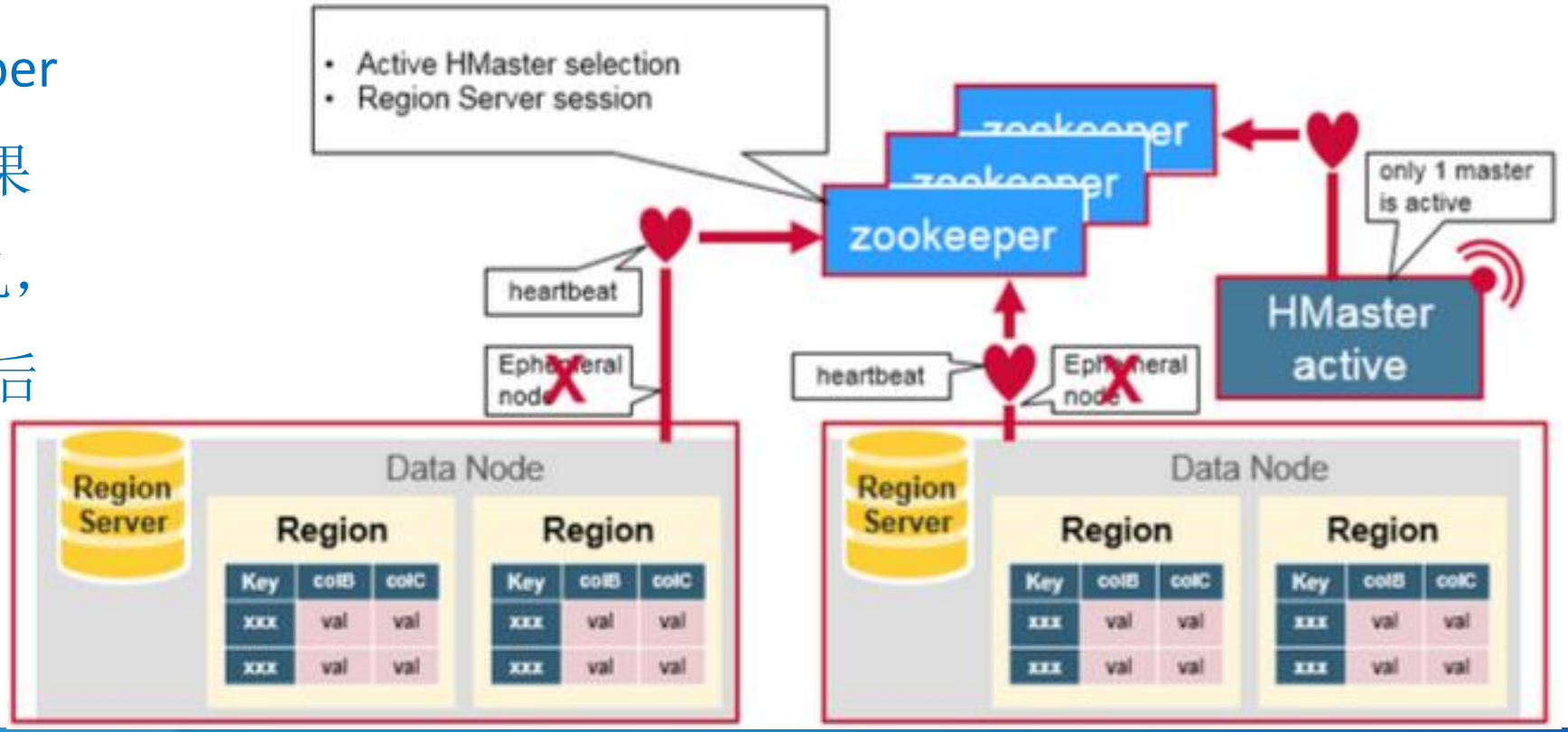
HBase架构-Zookeeper

- Zookeeper是一个分布式的协调服务。
- ZK管理着HMaster和HRegionServer的状态(available/alive等) 。
- ZK提供了宕机时通知功能，从而实现Hmaster的failover机制， RegionServer的failover机制。



HBase架构-Zookeeper

- ✓ 在HMaster和HRegionServer连接到ZooKeeper后创建Ephemeral节点，并使用Heartbeat机制维持这个节点的存活状态。
- ✓ Hmaster的监控zookeeper的ephemeral节点来监控RS的存活状态(默认：/hbase/rs/*)。
- ✓ 备用HMaster监控Zookeeper中的ephemeral节点，如果Active状态的HMaster宕机，且备用HMaster收到通知后切换为Active状态。



Hbase shell

运行命令：hbase shell进入Hbase shell console。

1.查看有哪些表
list

2.创建表：
语法： create <table>, {NAME => <family>, VERSIONS => <VERSIONS>}
Example:
create 'fanData',{NAME=>'INFO',VERSIONS => 1}

3.删除表
先disable表，在drop表。
disable 'fanData‘
drop 'fanData‘

4.查看表的结构
语法： describe <table>
Example:
describe 'fanData'

Hbase shell

5.修改表结构
语法： alter 't1', {NAME => 'f1'}, {NAME => 'f2', METHOD => 'delete'}
Example:
alter 'fanData',{NAME=>'INFO1',VERSIONS=>2}
enable 'fanData‘

6.添加数据
语法： put <table>,<rowkey>,<family:column>,<value>,<timestamp>
Example:
put 'fanData','1_WT02287_WS_20170502','INFO:1','8.33‘

7.查询数据-查询某行记录
语法： get <table>,<rowkey>,[<family:column>,....]
Example:
get 'fanData','1_WT02287_WS_20170502','INFO:1'
get 'fanData','1_WT02287_WS_20170502',{COLUMN=>'INFO:1'}
get 'fanData','1_WT02287_WS_20170502',{COLUMN=>'INFO:1',TIMESTAMP=>1493691362804}

Hbase shell

8.查询数据-扫描表
语法： scan <table>, {COLUMNS => [<family:column>,....], LIMIT => num}
Example:
scan 'fanData',{LIMIT=>2}
scan 'fanData',{COLUMNS=>'INFO:1',LIMIT=>2,STARTROW=>'1_WT02287_WS_20170501'}
scan 'fanData',{COLUMNS=>'INFO',LIMIT=>2,STARTROW=>'1_WT02287_WS_20170501'}
scan 'fanData',{COLUMNS=>'INFO',LIMIT=>2,TIMERANGE=>[1493691362801,1493691362805]}
scan 'fanData',{FILTER=>"(PrefixFilter('1_WT'))"}

9.查询表中的行数
语法： count <table>, {INTERVAL => intervalNum, CACHE => cacheNum}
Example:
count 'fanData', {INTERVAL => 100, CACHE => 500}

10.删除数据
语法： deleteall <table>, <rowkey>, <family:column> , <timestamp>， 可以不指定列名，删除整行数据
语法： truncate <table>
Example:
deleteall 'fanData','1_WT02287_WS_20170502‘

THANKS