



QINGCLOUD 青云

# 大数据快速入门培训

08/13/2016

# HBase 大纲

- ▶ HBase 介绍及特点
- ▶ HBase 系统架构
- ▶ HBase 集群搭建
- ▶ HBase 存储结构
- ▶ HBase 关键流程
- ▶ HBase 使用及开发

# HBase 起源

- ▶ Google 三大论文中的BigTable
- ▶ 2007年Powerset 上最早应用
- ▶ 2008年成为Hadoop的一个子项目，放于contrib目录下
- ▶ Top-level-project

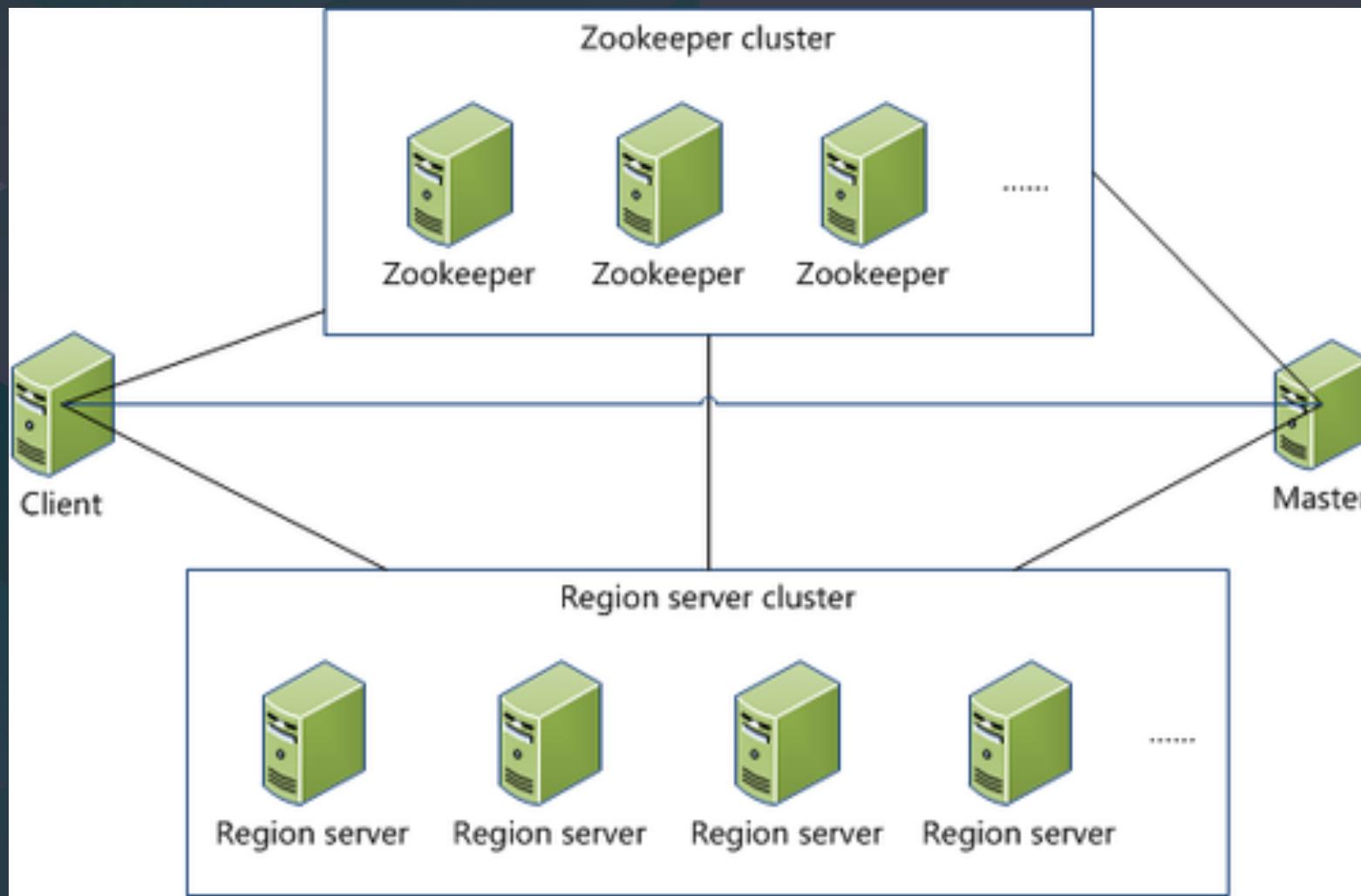
# HBase 简介

- ▶ NoSQL数据库，面向半结构化数据的存储和处理，低写入 / 查询延迟的系统
- ▶ 高可靠性、高性能的、列式存储的，可伸缩的，分布式系统。
- ▶ 基于row的强一致性访问
- ▶ 存储依赖于HDFS
- ▶ 表可以上亿行，百万列

# RDBMS VS HBase

	<b>RDBMS</b>	<b>HBase</b>
<b>Data layout</b>	Row-oriented	Column-family-oriented
<b>Transactions</b>	Multi-row ACID	Single row only
<b>Query language</b>	SQL	get/put/scan/etc *
<b>Security</b>	Authentication/Authorization <small><a href="http://blog.csdn.net/">http://blog.csdn.net/</a></small>	Work in progress
<b>Indexes</b>	On arbitrary columns	Row-key only
<b>Max data size</b>	TBs	~1PB
<b>Read/write throughput limits</b>	1000s queries/second	Millions of queries/second

# HBase 系统架构



# HBase 系统架构

- ▶ Client
  - ▶ 包含访问HBase的接口，client维护着一些cache来加快对HBase的访问，比如region的位置信息。

# HBase 系统架构

- ▶ Zookeeper
  - ▶ 保证任何时候，集群中只有一个master
  - ▶ 存储所有Region的寻址入口。
  - ▶ 实时监控Region Server的状态，将Region server的上线和下线信息实时通知给Master
  - ▶ 存储HBase的schema,包括有哪些table，每个table有哪些column family

# HBase 系统架构

- ▶ Master
  - ▶ 为Region server分配region
  - ▶ 负责region server的负载均衡
  - ▶ 发现失效的region server并重新分配其上的region
  - ▶ HDFS上的垃圾文件回收
  - ▶ 处理schema更新请求

# HBase 系统架构

- ▶ RegionServer
  - ▶ RegionServer维护Master分配给它的region，处理对这些region的IO请求
  - ▶ RegionServer负责切分在运行过程中变得过大的region

# HBase 集群搭建

- ▶ 准备程序包，包括jdk、hadoop、hbase，兼容关系参看[官网](#)
- ▶ 完成相关配置
- ▶ 启动进程并验证集群状态

# HBase 程序包准备

- ▶ 安装JDK 1.7以上版本，配好环境变量
- ▶ 下载稳定版本的hbase bin.tar.gz 或src.tar.gz(tar -xzvf 解压)
- ▶ (可选) 下载的src，解压后手动编译
  - ▶ 下载并安装mvn
  - ▶ tar -xzvf hbase-<version>-src.tar.gz;cd hbase-<version>
  - ▶ 修改pom中<hadoop-two.version>中hadoop版本为当前hbase依赖的hdfs版本
  - ▶ 执行 mvn -DskipTests package assembly:single install
  - ▶ 在./hbase-assembly/target/下有hbase-<version>-bin.tar.gz

# Hadoop 程序包准备

- ▶ 下载稳定版本的hadoop src.tar.gz 手动编译（native）
- ▶ 安装protobuf 2.5.0 (rpc, 序列化)
  - ▶ 需要安装libtool,autoconf
- ▶ 工具：gcc, cmake, ant, git, findbugs等等
- ▶ Lzo和snappy压缩包
- ▶ mvn package -Pdist,native -DskipTests -Dtar -Drequire.snappy -Dbundle.snappy -Dsnappy.lib=/usr/local/lib
- ▶ ./hadoop-dist/target/hadoop-<version>.tar.gz



# HBase 环境配置

- ▶ 主节点无秘钥登录，为了执行总控脚本start\_hbase.sh , stop\_hbase.sh
  - ▶ 主节点~/.ssh/id\_rsa.pub >> 到从节点及主节点的~/.ssh/authorized\_keys
- ▶ 配置集群所有节点host /etc/hosts
  - ▶ ip hostname
- ▶ 系统环境/etc/security/limits.conf
  - ▶ 修改启动用户的nofile和nproc值可为65535

# HBase 参数配置

- ▶ hbase-env.sh
  - ▶ JAVA\_HOME, HBASE\_LOG\_DIR, HBASE\_PID\_DIR, HBASE\_CONF\_PATH, HBASE\_HOME
  - ▶ HBASE\_LIBRARY\_PATH(native位置)
  - ▶ export HBASE\_MANAGES\_ZK=false
  - ▶ 内存配置
    - HBASE\_HEAPSIZE
    - HBASE\_MASTER\_OPTS
    - HBASE\_REGIONSERVER\_OPTS

# HBase 参数样例

```
#!/usr/bin/env bash

export JAVA_HOME=/usr/jdk
export HBASE_LOG_DIR=/data/hbase/logs
export HBASE_PID_DIR=/data/hbase/pids
export HBASE_CONF_PATH=/usr/local/hbase/conf
export HBASE_HOME=/usr/local/hbase
export HADOOP_HOME=/usr/local/hadoop
export JAVA_LIBRARY_PATH=$HADOOP_HOME/lib/native
export HBASE_LIBRARY_PATH=$HBASE_LIBRARY_PATH:$JAVA_LIBRARY_PATH
export HBASE_MANAGES_ZK=false
export HBASE_HEAPSIZE=4G
export HBASE_REGIONSERVER_OPTS="-Xmx4096m -Xms4096m -Xmn128m -
    XX:+UseParNewGC -XX:+UseConcMarkSweepGC -XX:CMSInitiatingOccupancyFraction=70
    -verbose:gc -XX:+PrintGCDetails -XX:+PrintGCTimeStamps -Xloggc:$HBASE_LOG_DIR/gc-
    regionserver.log
```

# HBase 参数配置

- ▶ hbase-site.xml
  - ▶ hbase.cluster.distributed true
  - ▶ hbase.rootdir hdfs://namenode:9000/hbase
  - ▶ hbase.zookeeper.quorum 172.17.101.2
  - ▶ zookeeper.znode.parent /hbase/hbs-nxwk31gl
  - ▶ zookeeper.session.timeout 60000
  - ▶ hbase.regionserver.codecs snappy,lzo
- ▶ regionservers
  - ▶ 配置要启动regionserver服务器ip
- ▶ backup-masters
  - ▶ 配置backup master

# HDFS 启动

- ▶ 配置参照hadoop，主要是fs.defaultFS配好。
- ▶ 配置etc/hadoop/slaves 将datanode节点写入
- ▶ bin/hadoop namenode -format 格式化HDFS
- ▶ 执行sbin/start-dfs.sh 启动所有进程namenode, secondarynamenode, datanode
- ▶ 验证启动成功 bin/hadoop dfsadmin -report
- ▶ 停止：sbin/stop-dfs.sh

# HBase 服务

- ▶ 先启动zookeeper和hdfs
- ▶ `bin/start-hbase.sh` `bin/stop-hbase.sh`  
`bin/hbase-daemon.sh start master`  
`bin/hbase-daemon.sh start regionserver`
- ▶ 检查集群状态，可通过主节点16010端口查看  
`bin/hbase hbck`

# HBase 存储结构

- ▶ HBase 表结构
- ▶ HBase 物理存储

# HBase 表结构

RowKey	column-family-1		column-family-2			column-family-3
	column-A	column-B	column-A	column-B	column-C	column-A
key001	t2:hk t1:jy		t4:ipad t3:ipod t1:iphone			
key002	t3:style t1:wk		t3:smile t2:jk	t2:wtf	t1:hw	
key003		t1:high				t4:powerful

SortedMap(RowKey, List(SortedMap(Column, List(Value, Timestamp)))) )

# HBase 表结构

- ▶ Row key
  - ▶ 用来检索记录的主键，访问table中的行方式：
    - ▶ 通过单个row key访问
    - ▶ 通过row key的range进行scan
    - ▶ 全表扫描
  - ▶ 可以是任意字符串（字节数组），存储时，数据按照row key字典序排序存储，设计key时应将经常读取的行存储到一起

# HBase 表结构

- ▶ 列族和列
  - ▶ 列族是表的scheme的一部分（而列不是），必须在使用表之前定义，列名是以列族为前缀的。列可动态扩展，无需预先定义。

# HBase 表结构

- ▶ Cell和Timestamp
  - ▶ Cell：通过rowkey和column（family + label）确定的叫做cell，每个cell保存多个version，version通过timestamp来做索引。
  - ▶ Timestamp：64位整型，默认是写入时的系统时间（精确到毫秒），也可以自己生成，时间倒叙排序
- ▶ 回收方式
  - ▶ 最新的n个version
  - ▶ 最近一段时间的version

# HBase 表操作

```
$ cd /usr/local/hbase
```

```
$ bin/hbase shell
```

```
hbase(main):001:0> create 'test', 'cf'  
0 row(s) in 1.2130 seconds
```

```
=> Hbase::Table - test  
hbase(main):002:0> list 'test'  
TABLE  
test  
1 row(s) in 0.0180 seconds  
  
=> ["test"]
```

# HBase 表操作

```
hbase(main):003:0> put 'test', 'row1', 'cf:a', 'value1'  
0 row(s) in 0.0850 seconds
```

```
hbase(main):004:0> put 'test', 'row2', 'cf:b', 'value2'  
0 row(s) in 0.0110 seconds
```

```
hbase(main):005:0> put 'test', 'row3', 'cf:c', 'value3'  
0 row(s) in 0.0100 seconds
```

```
hbase(main):006:0> scan 'test'  
ROW  
row1  
row2  
row3  
3 row(s) in 0.0230 seconds  
COLUMN+CELL  
column=cf:a, timestamp=1469163844008, value=value1  
column=cf:b, timestamp=1469163862005, value=value2  
column=cf:c, timestamp=1469163899601, value=value3
```

# HBase 表操作

```
hbase(main):007:0> get 'test', 'row1'  
COLUMN          CELL  
cf:a           timestamp=1469094709015, value=value1  
1 row(s) in 0.0350 seconds
```

```
hbase(main):008:0> disable 'test'  
0 row(s) in 1.1820 seconds
```

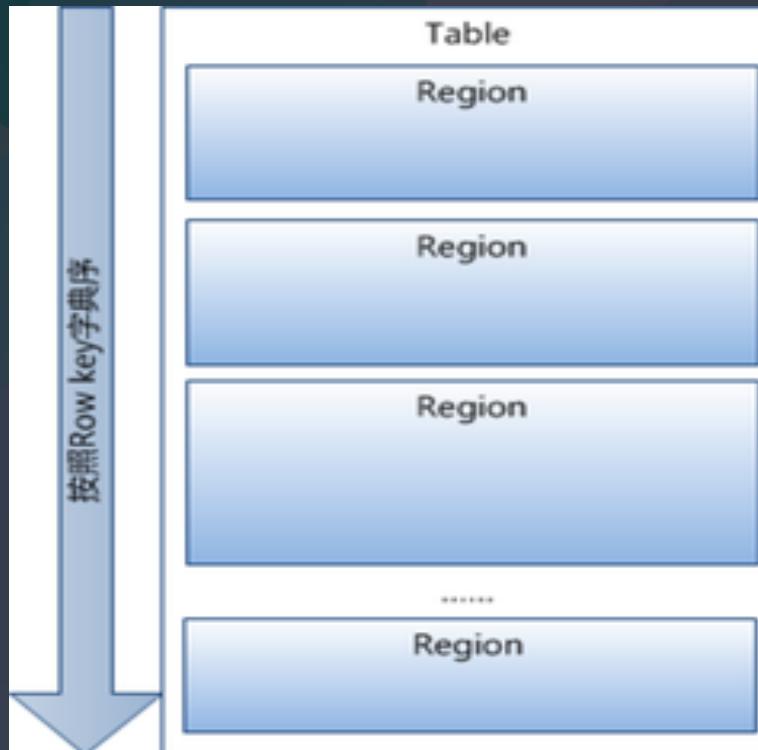
```
hbase(main):009:0> drop 'test'  
0 row(s) in 0.1370 seconds
```

# HBase Table 设计原则

- ▶ region大小介于10G-50G
- ▶ cell 大小不要超过10M，否则可以考虑数据存储在HDFS上，而在HBase上存储指针
- ▶ 一个table的column family不宜过多，在1到3个之间
- ▶ 一个table的region数目在50-100个比较好。特例：有大量冷数据
- ▶ column family 名字越短越好

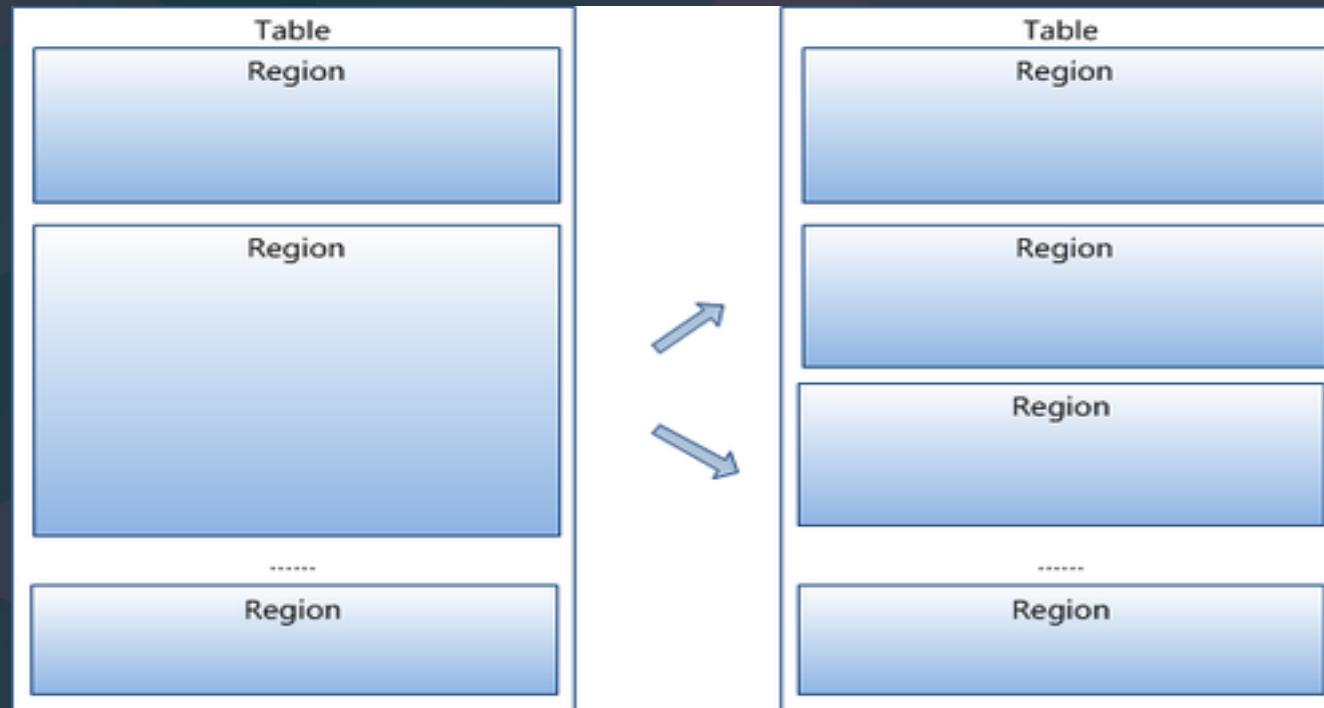
# HBase 物理存储

- ▶ Table所有行按照row key字典序排列，在行的方向上分割为多个Region



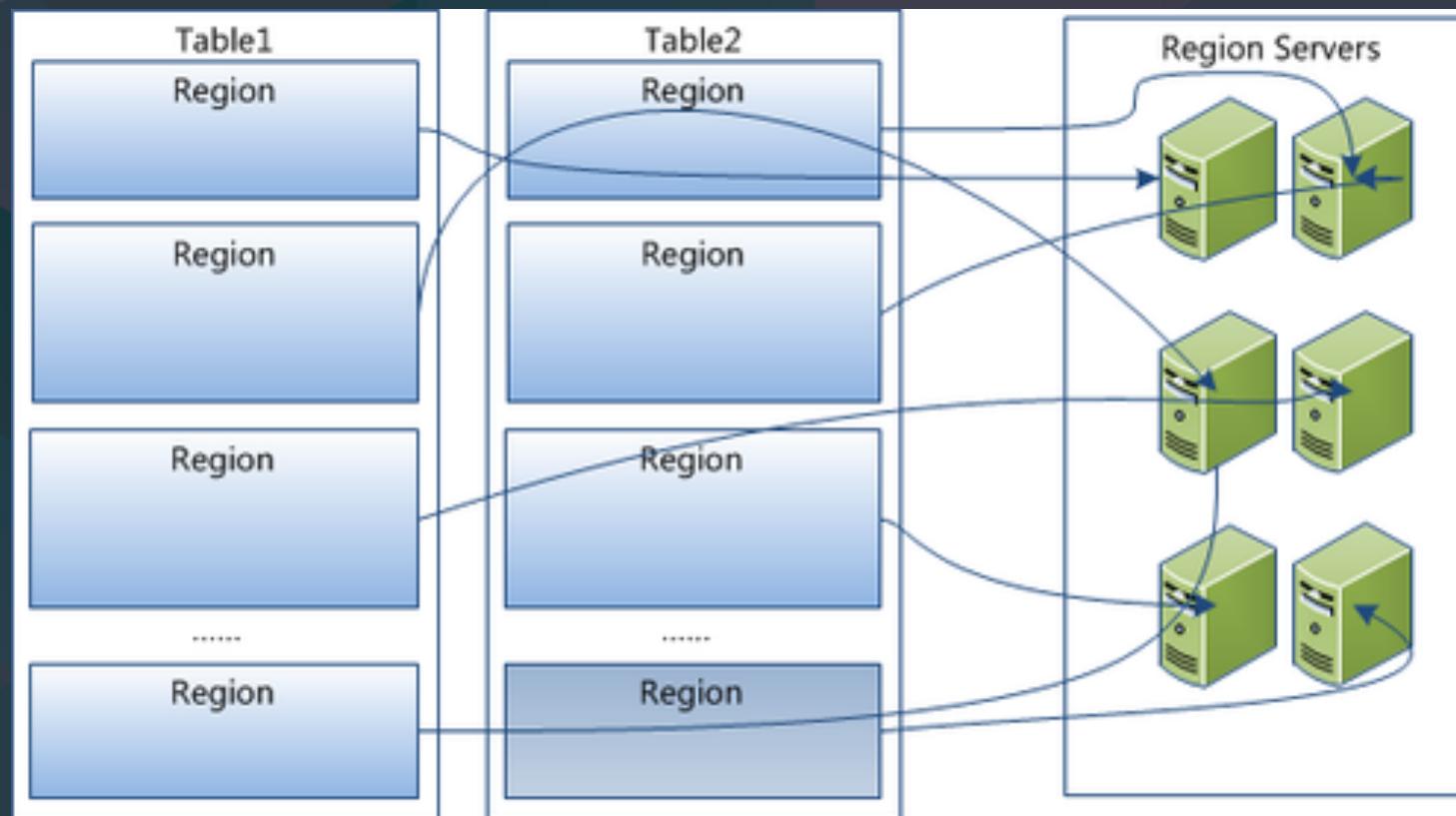
# HBase 物理存储

- ▶ 一个表最开始只有一个region，随着数据增多，region不断增大，达到一定阈值时，会split为2个region

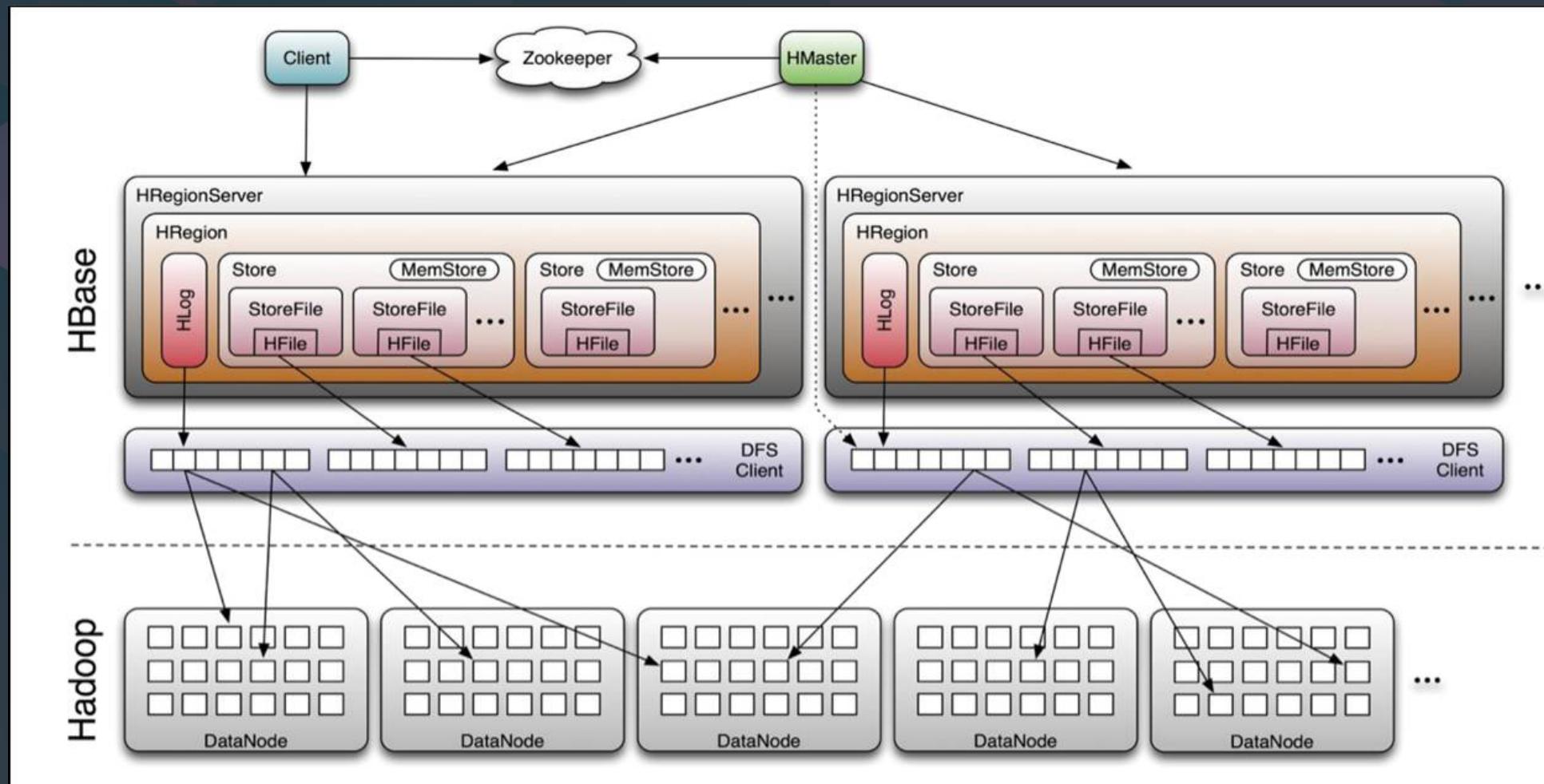


# HBase 物理存储

- Region是HBase中负载均衡的最小单元。



# HBase 物理存储



# HBase 物理存储

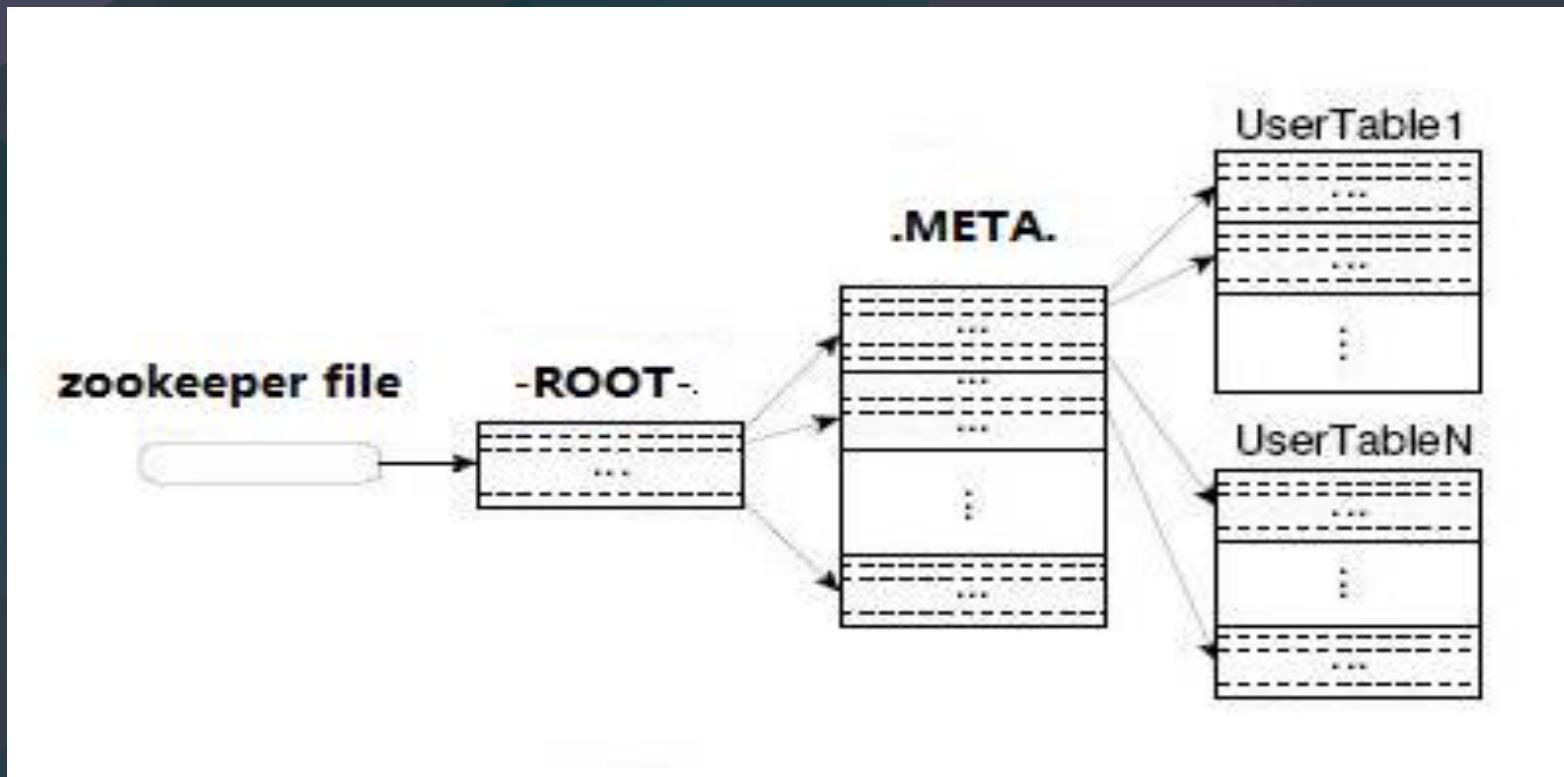
- ▶ Table
  - ▶ Region
    - ▶ Store
      - ▶ MemStore
      - ▶ StoreFile
        - ▶ Block
- ▶ /hbase
  - ▶ <Table>
  - ▶ <Region>
  - ▶ <ColumnFamily>
  - ▶ <StoreFile>

# HBase 物理存储

- ▶ Region并不是存储的最小单元。事实上，Region由一个或者多个Store组成，每个store保存一个columns family。每个Store又由一个memStore和0至多个StoreFile组成。StoreFile以Hfile格式保存在HDFS上。
- ▶ Hlog ( Write ahead log ) 用做灾难恢复，每个regionserver维护一个。
- ▶ BlockCache 读缓存，默认是LRU策略，一个regionserver一个。

# HBase 关键流程

## ► Region 定位

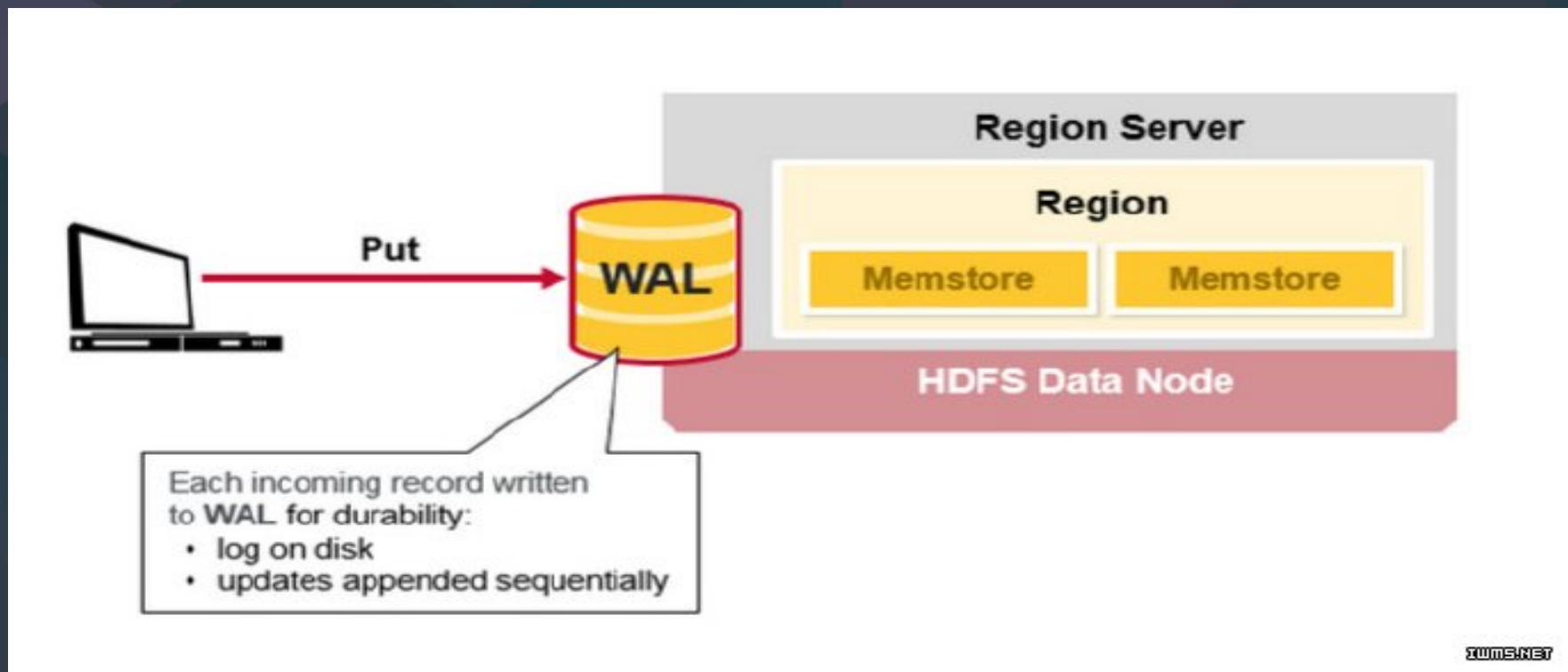


# HBase 关键流程

- ▶ Region 定位
  - ▶ 目的是找到rowkey ( 或者rowkey range ) 所在的region
  - ▶ 三层 :
    - ▶ zookeeper中保存-root- region所在的regionserver
    - ▶ -root- region 中保存.Meta. Region所在的regionserver
    - ▶ .Meta. Region保存了HBase中所有region的位置信息

# HBase 关键流程

## ▶ 写操作



# HBase 关键流程

- ▶ 写操作
  - ▶ 定位regionserver，先写WAL，根据tablename和rowkey定位region，根据列族定位Store，写memstore，随写随排序，达到阀值，创建一个新的memstore，同时将老的memstore放到队列，flush到硬盘形成StoreFile
  - ▶ StoreFile是只读的，多个小的storeFile进行minor compact，storeFile数量达到阀值会进行major compact形成一个大的StoreFile，当StoreFile过大，又会进行region的split

# HBase 关键流程

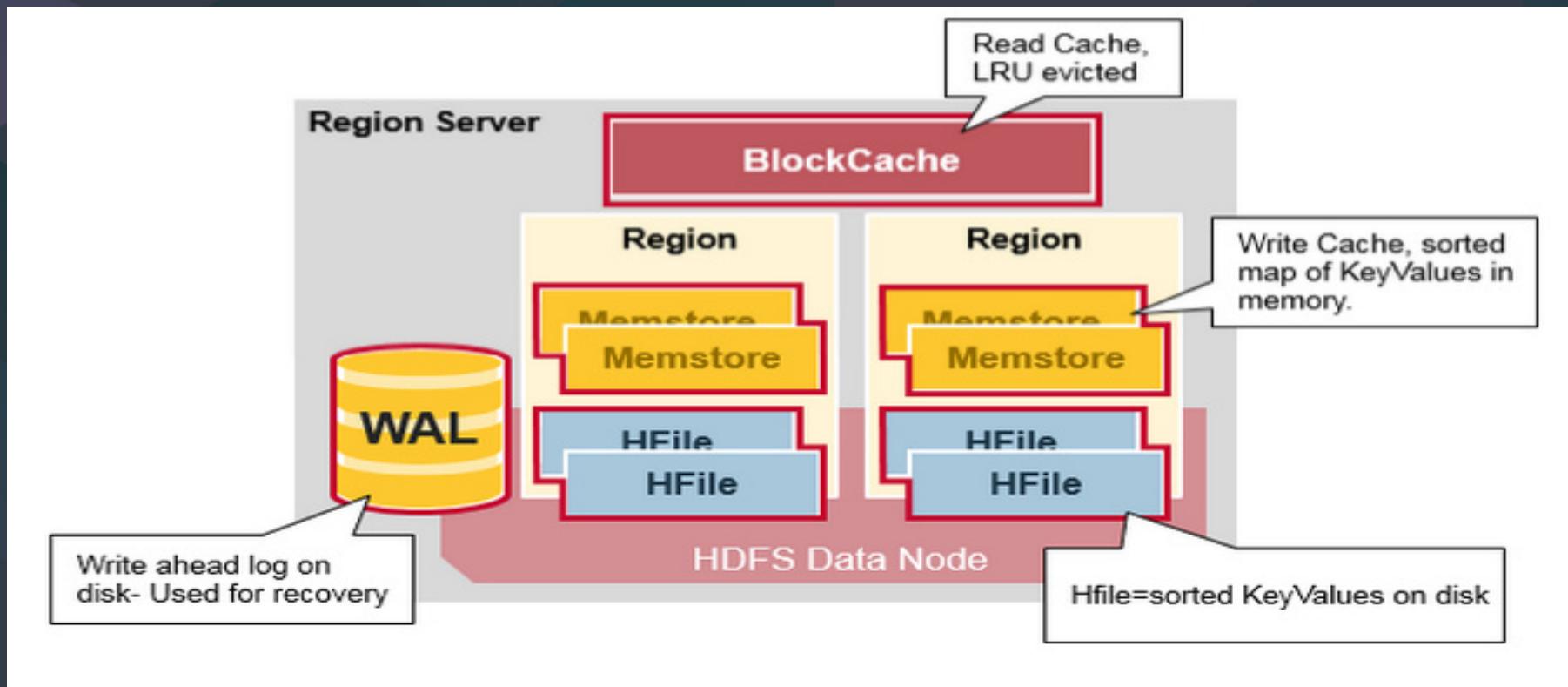
## ▶ Memstore Flush

最小flush单元是“region”，而不是memstore，触发条件：

- ▶ 一个MemStore达到`hbase.hregion.memstore.flush.size`
- ▶ 全部MemStore内存大小达到`hbase.regionserver.global.memstore.size`， flush按照由达到小的顺序进行，直到小于`hbase.regionserver.global.memstore.size.lower.limit`
- ▶ WAL 达到`hbase.regionserver.max.logs`，按时间由老到新顺序flush memstore

# HBase 关键流程

## ▶ 读操作



# HBase 关键流程

- ▶ 读操作
  - ▶ 定位Regionserver，根据tablename和rowkey定位region，根据列族定位Store，优先memstore，再找blockcache，最后是全部storefile（Hfile）
  - ▶ 如果到了Hfile（分为多个块，默认64k），则会整块读到BlockCache中。（连续的rowkey）
  - ▶ 将结果合并

# HBase 关键流程

- ▶ 删除操作
  - ▶ 删除version数据加墓碑标记确保读取时不读取出来，直到下次major compact时真正删除

# HBase 操作优化

- ▶ Put操作如果追求效率可以关闭WAL
- ▶ Put操作关掉autoFlush，提高插入性能
- ▶ 离线Major Compact
- ▶ 启用压缩
  - ▶ HBase> create 'test2', { NAME => 'cf2', COMPRESSION => 'SNAPPY' }
  - ▶ HBase> disable 'test'  
HBase> alter 'test', {NAME => 'cf', COMPRESSION => 'GZ'}  
HBase> enable 'test'

# HBase 开发

- ▶ 建立连接，增删改查
- ▶ 提供了大量的内置filter ( rowkey , prefix等 ) ，过滤掉的数据不会被传回客户端。
- ▶ 协处理器coprocessor
  - ▶ 触发器：preget , preput
  - ▶ 服务器端的avg , count , min , max等

# HBase 开发

## ▶ 自带的性能测试

PerformanceEvaluation: 随机写、顺序写、increment、append、随机读、顺序读、scan等。

- ▶ \$ bin/hbase pe --nomapred --rows=10000 --presplit=10 randomWrite 10
- ▶ \$ bin/hbase pe --nomapred --rows=10000 --presplit=10 sequentialWrite 10
- ▶ YCSB

# HBase 开发

- ▶ 使用MapReduce导入数据有三种方案：
  - ▶ MapReduce使用HBase提供的JAVA API从HDFS 导入到HBase。
    - ▶ \$ bin/hadoop jar \$APP\_HOME/hbase-tools-1.0.0.jar com.qingcloud.hbase.ImportByMR /user/inputPath
  - ▶ MapReduce将HDFS中数据转化为HFile 格式，再BulkLoad到HBase。
    - ▶ \$ bin/hadoop jar test\_hbase/hbase\_tools-1.0.0.jar com.qingcloud.hbase.ImportByBulkLoad /user/inputPath /user/outputPath
    - ▶ \$ bin/hadoop jar /usr/local/hbase/lib/hbase-server-<VERSION>.jar completebulkload /user/outputPath test\_import
  - ▶ 使用HBase ImportTsv 工具将格式化的HDFS数据导入到HBase。
    - ▶ bin/hbase org.apache.hadoop.hbase.mapreduce.ImportTsv -Dimporttsv.columns=HBASE\_ROW\_KEY,content:a -Dimporttsv.bulk.output=/user/outputPath test\_import /user/inputPath

# HBase 开发

```
$ cd /usr/local/hbase
```

```
$ bin/hbase shell
```

```
hbase(main):001:0> create 'movie_rating', 'a'
```

```
$ cat ratings.csv | awk -F ',' '{print $2}' | sort | uniq | wc -l
```

```
$ bin/hadoop fs -put /usr/local/ml-latest/ratings.csv /inputPath
```

```
$ bin/hadoop jar ..../hbase/hbase-tools-1.0.0.jar com.qingcloud.hbase.ImportByBulkLoad  
/inputPath /outputPath
```

```
$ bin/hadoop jar /usr/local/hbase/lib/hbase-server-1.2.2.jar completebulkload /outputPath  
movie_rating
```

```
hbase(main):002:0> get 'movie_rating', '1'
```

```
hbase(main):003:0> scan 'movie_rating' ,{ LIMIT => 10}
```

```
hbase(main):004:0> count 'movie_rating', CACHE => 1000
```

```
hbase(main):005:0> scan 'movie_rating',{FILTER => "(PrefixFilter ('199'))" }
```

# HBase 使用场景

- ▶ 业务场景简单，不需要关系型数据库中的很多特性。
- ▶ 对行的一致性有要求，并发能力以及吞吐量、响应延迟有较高需求
- ▶ 有数据分析需求，和MapReduce良好的兼容性。
- ▶ 写入性能好，性能不随数据增长而下降，随机访问良好
- ▶ 服务能力随服务器增长而线性增长，读写无单点问题。
- ▶ 范围扫描需求

# 关注我们



QingCloud-IaaS



青云QingCloud

[www.qingcloud.com](http://www.qingcloud.com)



# Thank you.

[stark@yunify.com](mailto:stark@yunify.com)