

**1.**

**1.1**

**HDFS**

**1.2. HDFS**

**block**

**1.3.**

**NameNode**

**?**

**1.4. HDFS**

**Block Size**

**1.5.**

## 1.6. SecondaryNameNode

## 1.7.

## 1.8. Client

## 1.9. Hadoop

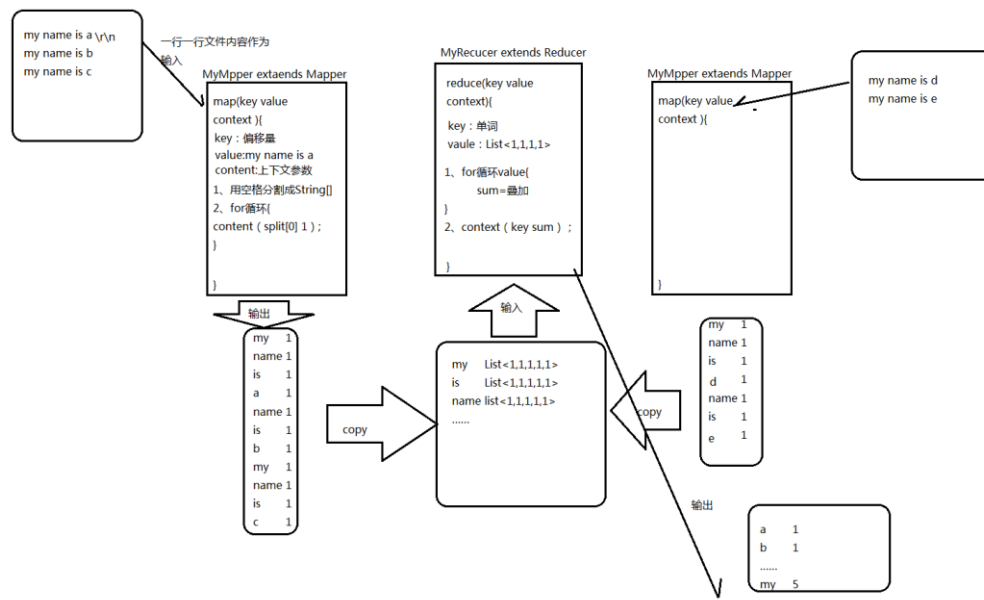
**2.**

## 2.1. Hadoop

## 2.2.

## 2.3. jps

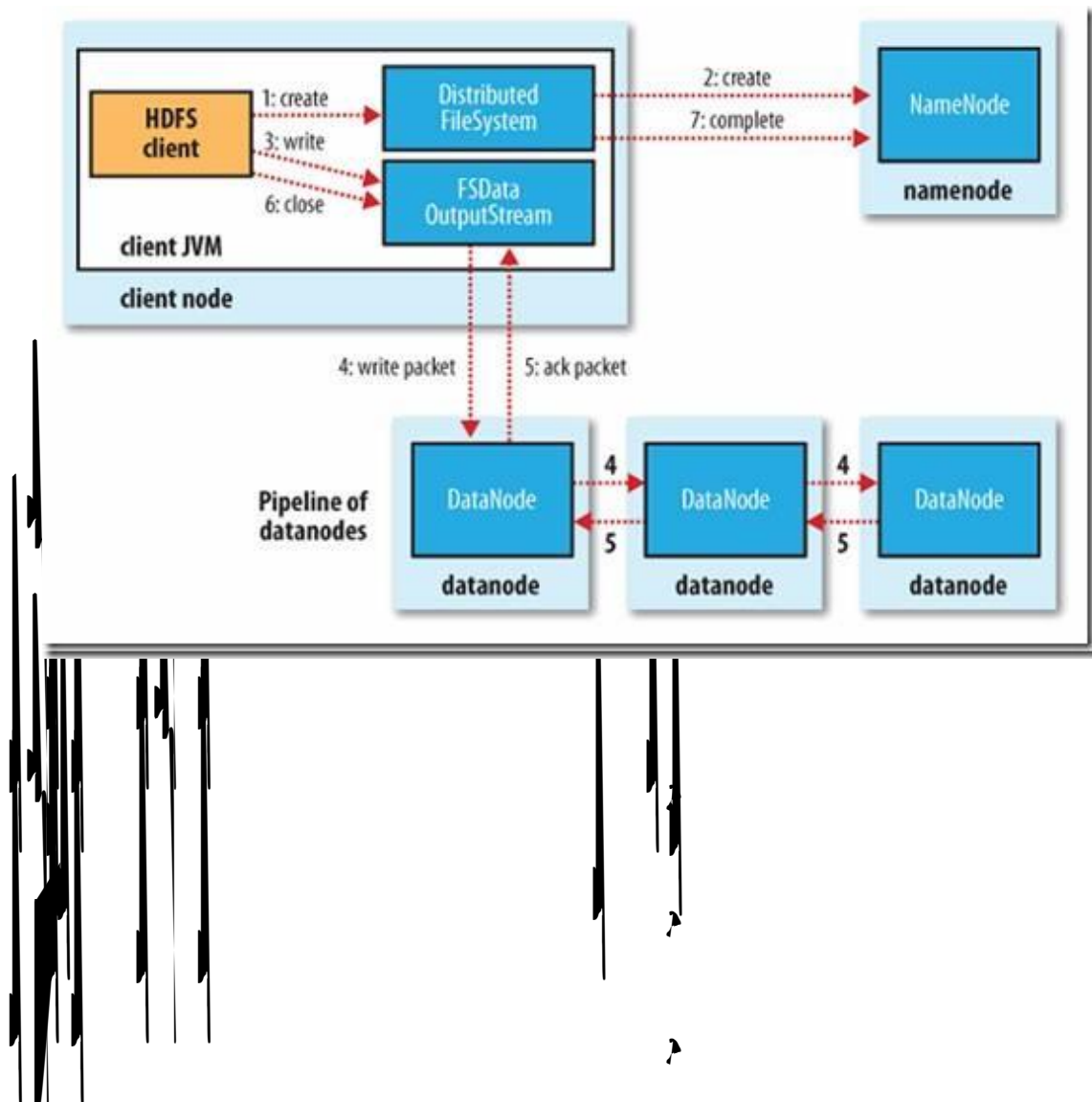
## 2.4. mapreduce ?



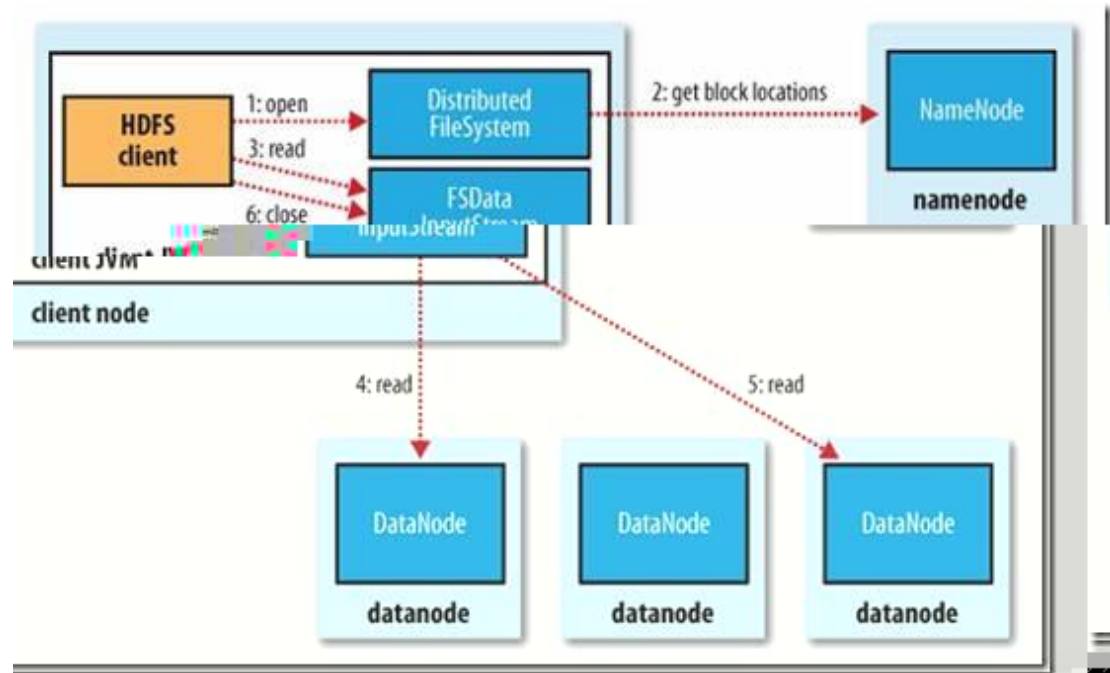
## 2.5. HDFS

?

### 2.5.1. hdfs



## 2.5.2.



## 2.6.

mapreduce

?

## 2.7. mapreduce

## 2.8. hadoop Combiner ?

## 2.9. hadoop

1. 简要描述如何安装配置一个 apache 开源版 hadoop。只描述即可，无需列出完整步骤，能列出步骤更好。

- 1.创建 hadoop 帐户。
- 2.setup.改 IP。
- 3.安装 java，并修改/etc/profile 文件，配置 java 的环境变量。
- 4.修改 Host 文件域名。
- 5.安装 SSH，配置无密钥通信。
- 6.解压 hadoop。
- 7.配置 conf 文件下 hadoop-env.sh、core-site.sh、mapre-site.sh、hdfs-site.sh。
- 8.配置 hadoop 的环境变量。

9.Hadoop namenode -format

91-hadoop namenode -format

10.Start-all

## 2.10. hadoop

2. 请列出正常工作的 Hadoop 集群中 Hadoop 都分别需要启动哪些进程，他们的作用分别是什么，尽可能写的全面些。

namenode：管理集群，并记录 datanode 文件信息。

Secondname:可以做冷备，对一定范围内数据做快照性备份。

Datanode:存储数据

Jobtracker :管理任务，并将任务分配给 tasktracker。

Tasktracker:任务执行方。

## 2.11.

3. 启动 Hadoop 时报如下错误，如何解决

```
ERROR org.apache.hadoop.hdfs.server.namenode.NameNode:
org.apache.hadoop.hdfs.server.common.InconsistentFSStateException: Directory
/tmp/hadoop-root/dfs/name is in an inconsistent state: storage directory does not exist or is not
accessible.
    at
    org.apache.hadoop.hdfs.server.namenode.FSImage.recoverTransitionRead(FSImage.java:303)
    at
    org.apache.hadoop.hdfs.server.namenode.FSDirectory.loadFSImage(FSDirectory.java:100)
    at
    org.apache.hadoop.hdfs.server.namenode.FSNamesystem.initialize(FSNamesystem.java:388)
    at
    org.apache.hadoop.hdfs.server.namenode.FSNamesystem.<init>(FSNamesystem.java:362)
    at
    org.apache.hadoop.hdfs.server.namenode.NameNode.initialize(NameNode.java:276)
    at org.apache.hadoop.hdfs.server.namenode.NameNode.<init>(NameNode.java:496)
    at
    org.apache.hadoop.hdfs.server.namenode.NameNode.createNameNode(NameNode.java:1279)
    at org.apache.hadoop.hdfs.server.namenode.NameNode.main(NameNode.java:1288)
```

## 2.12.

4. 请写出以下执行命令

- 1) 杀死一个 job
- 2) 删除 hdfs 上的/tmp/aaa 目录
- 3) 加入一个新的存储节点和删除一个计算节点需要刷新集群状态命令

`hadoop job -list` 拿到 job-id , `hadoop job -kill job-id`

`Hadoop fs -rmr /tmp/aaa`

加新节点时：

`Hadoop-daemon.sh start datanode`

`Hadoop-daemon.sh start tasktracker`

删除时：

`Hadoop mradmin -refreshnodes`

`Hadoop dfsadmin -refreshnodes`

## 2.13.      **hadoop**

5. 请列出你所知道的 hadoop 调度器，并简要说明其工作方法。



Fifo scheduler :默认，先进先出的原则

Capacity scheduler :计算能力调度器，选择占用最小、优先级高的先执行，依此类推。

Fair scheduler:公平调度，所有的 job 具有相同的资源。

## 2.14. mapreduce

6. 请列出在你以前的工作中所使用过的开发 map/reduce 的语言

## 2.15.

7. 当前日志采样格式为

a,b,c,d

b,b,f,e

a,a,c,f

请用你最熟悉的语言编写一个 map/reduce 程序，计算第四列每个元素出现的个数。

## 2.16.

8. 你认为用 Java, Streaming, pipe 方式开发 map/reduce，各有哪些优缺点。

## 2.17. hive

9. Hive 有哪些方式保存元数据的，各有哪些特点。

## 2.18. combiner partition

请简述 MapReduce 中 combiner, partition 作用

## 2.19. hive

## 2.20. hbase rowkey

## 2.21. mapreduce

```
public int getPartition(K key, V value,
                        int numReduceTasks) {
    return (key.hashCode() & Integer.MAX_VALUE) % numReduceTasks;
}
```

```
public int getPartition(K key, V value,
                        int numReduceTasks) {
    return ((key).hashCode()+value.hashCode() &
Integer.MAX_VALUE) % numReduceTasks;
}
```

2

```
public class HashPartitioner<K, V> extends Partitioner<K, V> {
    private int aa= 0;

    /** Use {@link Object#hashCode()} to partition. */
    public int getPartition(K key, V value,
                            int numReduceTasks) {
```

```
        return (key.hashCode()+(aa++) & Integer.MAX_VALUE) %  
numReduceTasks;  
    }
```

## 2.22. hadoop

### 2.22.1.

## 2.22.2.

查看 linux 的服务，可以关闭不必要的服务

停止打印服务

## 关闭 ipv6

## 调整文件最大打开数

```
* soft nofile 65535
* hard nofile 65535
* soft nproc 65535
* hard nproc 65535
```

## 修改 linux 内核参数

```
vi /etc/sysctl.conf
```

```
net.core.somaxconn = 32768
```

#web	listen	backlog	net.core.somaxconn
128	nginx	NGX_LISTEN_BACKLOG	511

**关闭 noatime**

**设置 readahead buffer**

**修改最大槽位数**

**调整心跳间隔**

**集群规模小于 300 时，心跳间隔为 300 毫秒**

**启动带外心跳**

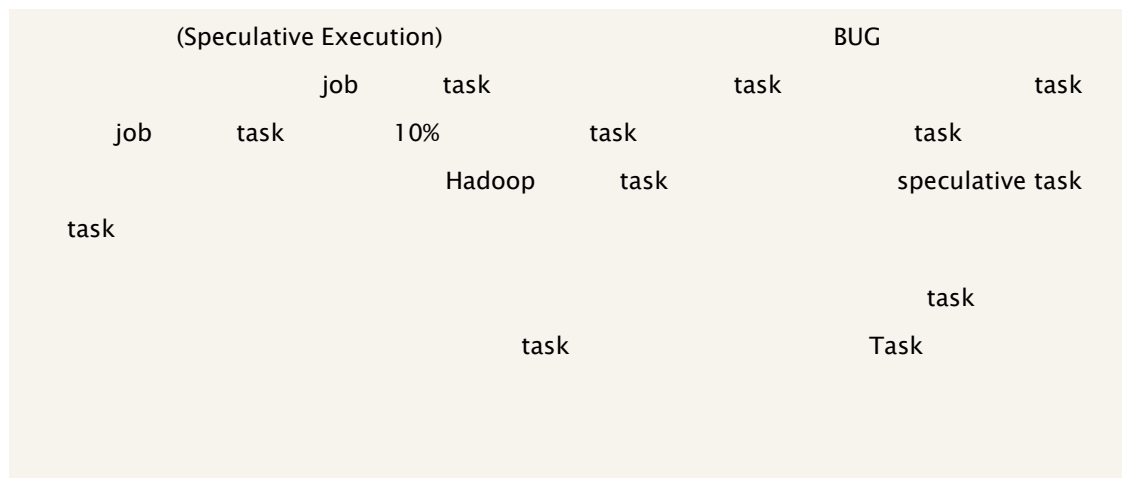
**配置多块磁盘**

**配置 RPC handler 数目**

**选择合适的压缩方式**



## 启用推测执行机制



## 设置是失败容忍度

启动 jvm 重用功能

设置任务超时时间

合理的控制 reduce 的启动时间

跳过坏记录

2.23.

job

reduce

2.24. datanode

2.25. combiner

2.26. hdfs

2.27. 3 datanode

datanode

2.28.

hadoop

**2.29.           hadoop**

**2.30.**

## 3. 15

### 3.1.

请选择您熟练掌握的 hadoop 版本，并基于此回答下列问题

☒ hadoop1.0    ☐ hadoop2.0

1. hadoop 的核心配置文件名称是什么？  
*core-site.xml*
2. "jps" 命令的用处？  
*查看 hadoop 节点进程*
3. 如何检查 namenode 是否正常运行？重启 namenode 的命令是什么？
4. 避免 namenode 故障导致集群宕机的解决方法是什么？
5. hbase 数据库对行键的设计要求是什么？

## Hadoop 面试

### 业务场景:

用户访问网站时, 每个页面会上报一条 pv 数据, 同时做一些业务操作, 会上报事件数据。  
如:

1. 用户浏览页面 (PV)
  2. 用户事件行为 (开户, 下单买基金.....)
  3. 页面 click 点击 (包含各种超链接, 可点击按钮: radio, checkbox.....)
- 每日 以上 3 项的上报数大致 10, 000, 000。

### 业务需求:

1. 需要按时间维度 (天, 周), 某种业务维度 (开户, 买基金...), 定时做统计(总人数, 金额等)。

例: 过去一周(隔日)的 pv, uv 数, 交易总金额。

2. 需要回溯历史数据, 如: 过去某个时间点 (段), 访问过某页面的用户, 在某个时间点 (段) 导致某种业务发生的统计数据。

例: 在 2015-01-01 至 2015-01-31 访问 A 页面, 并在 2015-01-01 至 2015-03-31 开户, 下单的用户数

### 技术方案:

请给出你的设计方案, 比如使用哪些技术框架、该框架起到的作用等。

答:

## 3.2. hive ,

3.3. storm ,

storm

3.4. datanode ,

3.5. Hbase , rowkey  
columnFamily , table

3.6. Redis, ,hbase,hive

3.7. shuffle ,

3.8. Mapreduce map reduce

,

### 3.9. `storm`

### 3.10. Hive

UDF,hive

### 3.11. Hadoop

### 3.12.

### 3.13.

#### 1) 设计日志收集分析系统

日志分布在各个业务系统中，我们需要对当天的日志进行实时汇总统计，同时又能查询历史的汇总数据（可以按PV、UV、IP等指标进行阐述）

#### 2) 如果你来做技术分享，你会选择什么主题，课程安排是什么样的？



5) Hive 语句实现 WordCount.

假设数据存放在 hadoop 下，路径为: /home/hadoop/worddata 里面全是一些单词

6) 给定 a、b 两个文件，各存放 50 亿个 url，每个 url 各占 64 字节，内存限制是 4G，找出 a、b 文件共同的 url？

7) 一个数据表存放 100 个最大数（步骤及算法复杂度）

8) 实时数据统计会用到哪些技术，它们各自的应用场景及区别是什么？



### 大数据岗位笔试题

1) String 与 StringBuffer 的区别，StringBuffer 与 StringBuilder 的区别

## 2) HashTable 和 HashMap, arrayList 和 vector, ArrayList 和 LinkedList 的异同

. E J  
- c b  
c b  
/ c b b  
E J E b J E b b ) E c b  
E J b b b E c b E J  
E c b E b b b b Hb  
J b c b E b b E c b . /  
E J E b E c b E J E c b  
E c b E J c b , b c  
a % d d E c b c b E c b % 8 %  
. ) b bdb % - 8 % / ) b bdb % 2 - 8 % 0 ) b  
bdb % - 8 % 1 ) b bdb % - 8 % 2 ) b bdb % - 8  
b b c b b b % 8  
c b % c J b b b % b % b b b % 8

!	!			!	b	b
						Hb

 c c b					 c b	
--	--	--	--	--	--	--

C:\WINDOWS\C:\WINDOWS\SYSTEM Hashtables				
Java				
Java.util.Properties	Hashtable	String keys	values	Properties
Hashtable				
Store()	Properties	Load()		
		Properties	keys	values
Properties		Hashtable	put()	String
keys	values	store()		String
Properties	store()	put()	get()	setProperty()
getProperty()	String			

3) 多线程实现方式 Thread 和 Runnable 的区别？

```

1. package org.thread.demo;
2. class MyThread extends Thread{
3.     private String name;
4.     public MyThread(String name) {
5.         super();
6.         this.name = name;
7.     }
8.     public void run(){
9.         for(int i=0;i<10;i++){
10.            System.out.println("        "+this.name+" , i="+i);

```

```
11. }
12. }
13. }
14. package org.thread.demo;
15. public class ThreadDemo01 {
16. public static void main(String[] args) {
17. MyThread mt1=new MyThread("    a");
18. MyThread mt2=new MyThread("    b");
19. mt1.run();
20. mt2.run();
21. }
22. }
```

```
1. package org.thread.demo;
2. public class ThreadDemo01 {
3. public static void main(String[] args) {
4. MyThread mt1=new MyThread("    a");
5. MyThread mt2=new MyThread("    b");
6. mt1.start();
7. mt2.start();
8. }
9. };
```

```
1. public interface Runnable{
2. public void run();
3. }
```

```

1. package org.runnable.demo;
2. class MyThread implements Runnable{
3.     private String name;
4.     public MyThread(String name) {
5.         this.name = name;
6.     }
7.     public void run(){
8.         for(int i=0;i<100;i++){
9.             System.out.println("          "+this.name+", i="+i);
10.        }
11.    }
12. };

```

```

1. package org.runnable.demo;
2. import org.runnable.demo.MyThread;
3. public class ThreadDemo01 {
4.     public static void main(String[] args) {
5.         MyThread mt1=new MyThread("    a");
6.         MyThread mt2=new MyThread("    b");
7.         new Thread(mt1).start();
8.         new Thread(mt2).start();
9.     }
10. }

```

- 
- 

```

1. package org.demo.dff;
2. class MyThread extends Thread{
3.     private int ticket=10;

```

```

4. public void run() {
5.     for(int i=0;i<20;i++){
6.         if(this.ticket>0){
7.             System.out.println("        ticket"+this.ticket--);
8.         }
9.     }
10. }
11. };

```

```

1. package org.demo.dff;
2. public class ThreadTicket {
3.     public static void main(String[] args) {
4.         MyThread mt1=new MyThread();
5.         MyThread mt2=new MyThread();
6.         MyThread mt3=new MyThread();
7.         mt1.start();//                10                30
8.         mt2.start();//                10
9.         mt3.start();//
10. }
11. }

```

```

1. package org.demo.runnable;
2. class MyThread implements Runnable{
3.     private int ticket=10;
4.     public void run(){
5.         for(int i=0;i<20;i++){
6.             if(this.ticket>0){
7.                 System.out.println("        ticket"+this.ticket--);
8.             }
9.         }
10.     }
11. }

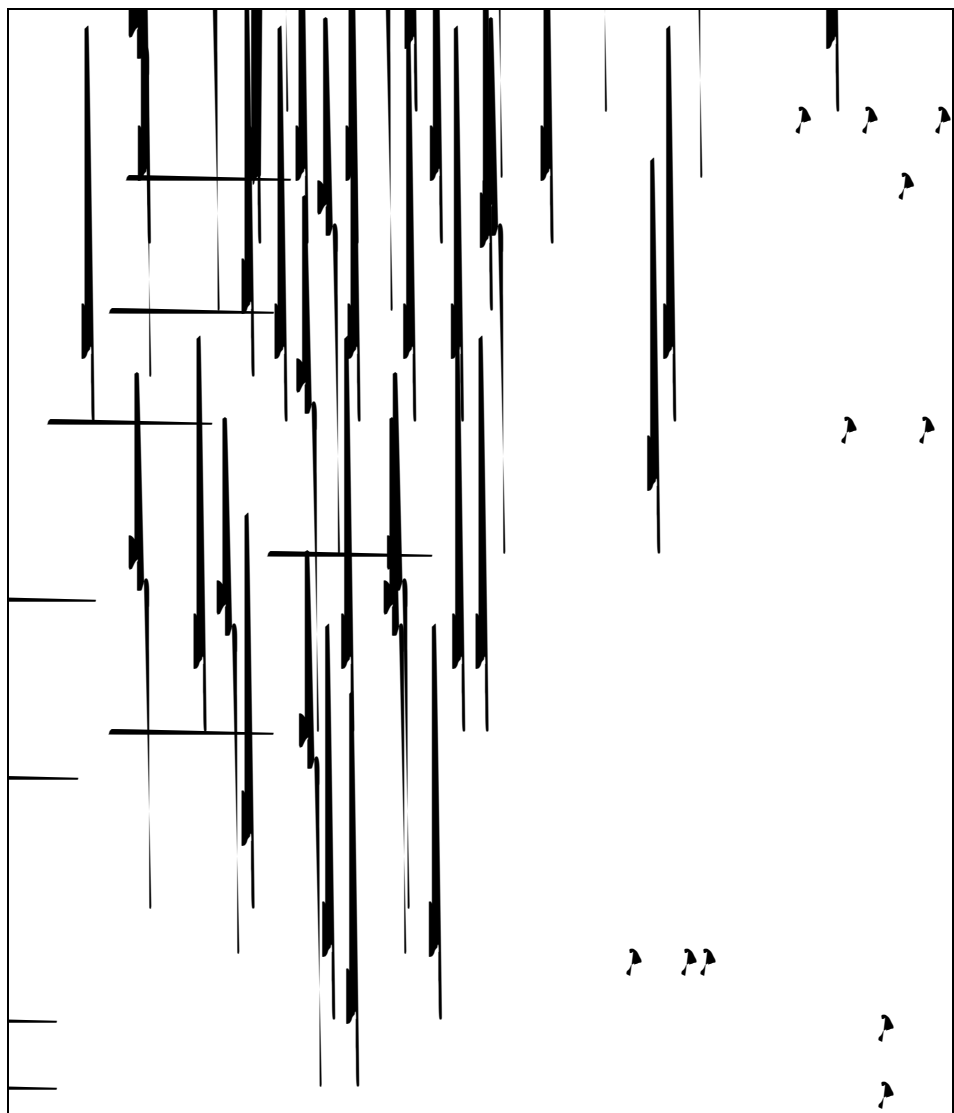
12. package org.demo.runnable;
13. public class RunnableTicket {
14.     public static void main(String[] args) {
15.         MyThread mt=new MyThread();
16.         new Thread(mt).start();//        mt        Thread
17.         new Thread(mt).start();//        mt
18.         new Thread(mt).start();
19.     }

```

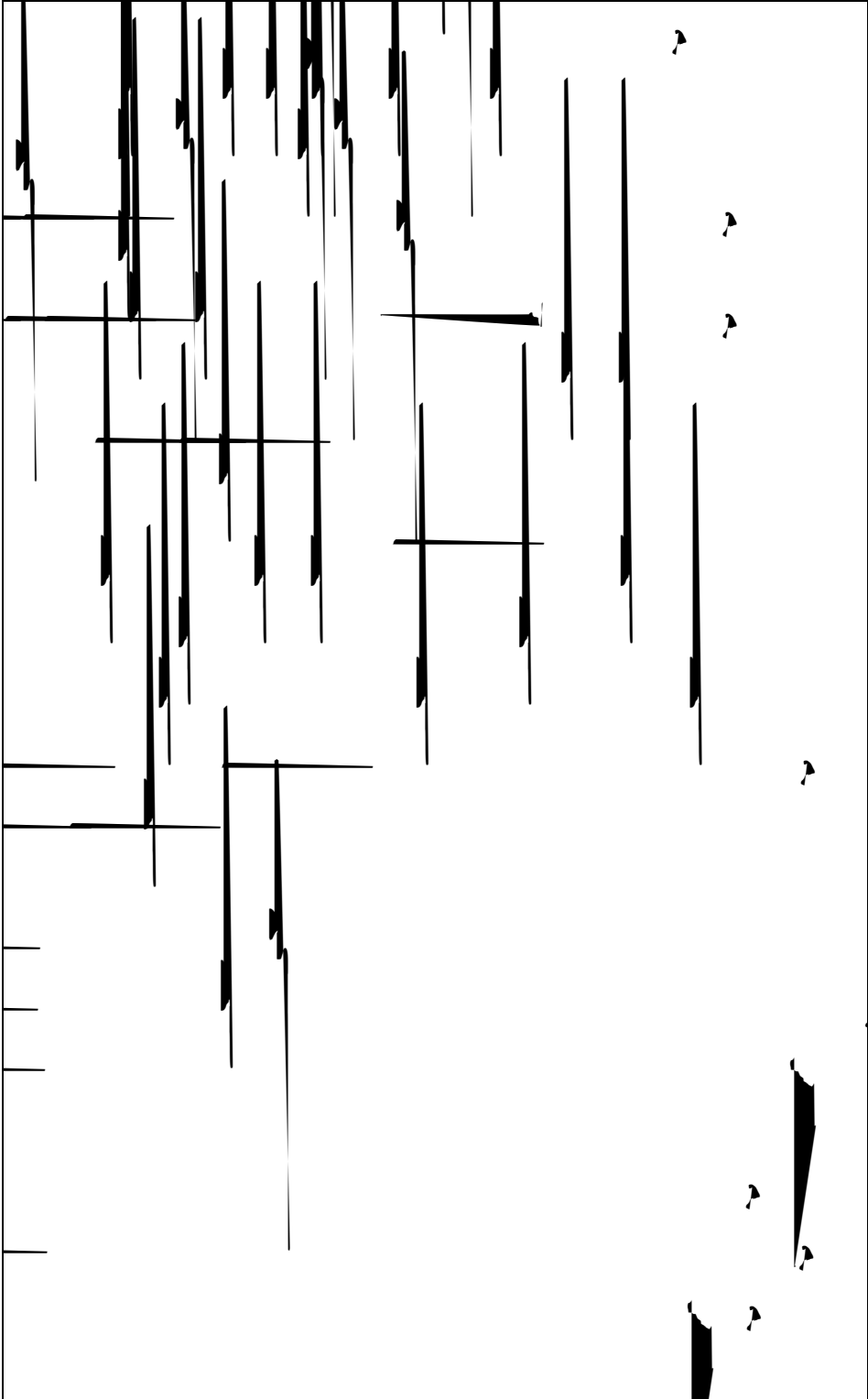
20. };

### 3.14.

1. 一个 HADOOP 环境，整合了 HBASE 和 HIVE，是否有必要给 HDFS 和 HBASE 都分别配置压缩策略？请给出对压缩策略的建议。







3. 简述 Hbase 性能优化的思路

4. 简述 Hbase filter 的实现原理是什么？结合实际项目经验，写出几个使用 filter 的场景

5. ROWKEY 的后缀匹配怎么实现？例如 ROWKEY 是 yyyyMMDD-UserID 形式，如 UserID 为条件查询数据，怎样实现。

6. 简述 Hive 中的虚拟列作用是什么，使用它的注意事项

7. 如果要存储海量的小文件（大小都是几百 K~几 M），请简述自己的设计方案

8. 有两个文本文件，文件中的数据按行存放。请编写 MapReduce 程序，找到两个文件中彼此不相同的行。（写出思路即可）

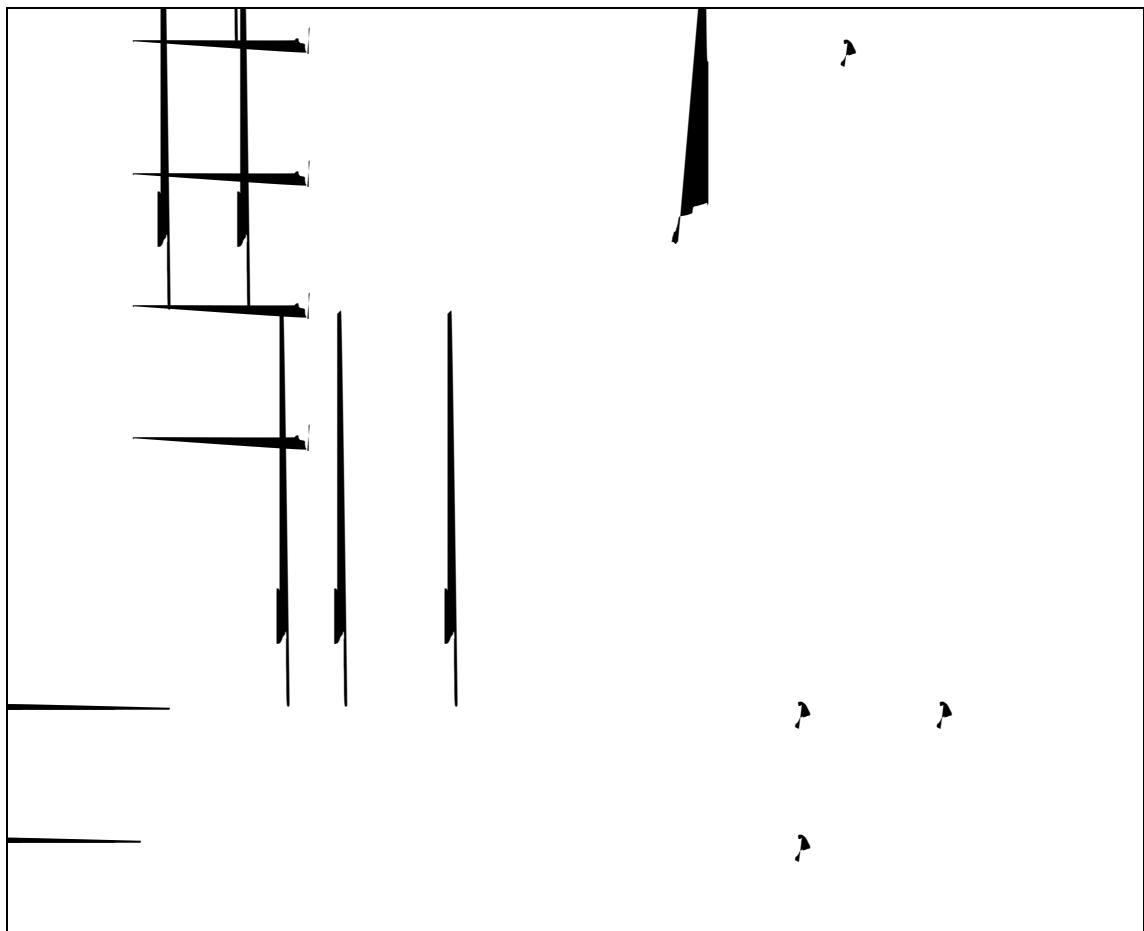
行中 找个 第几个字节做 key，其余字节做 value，

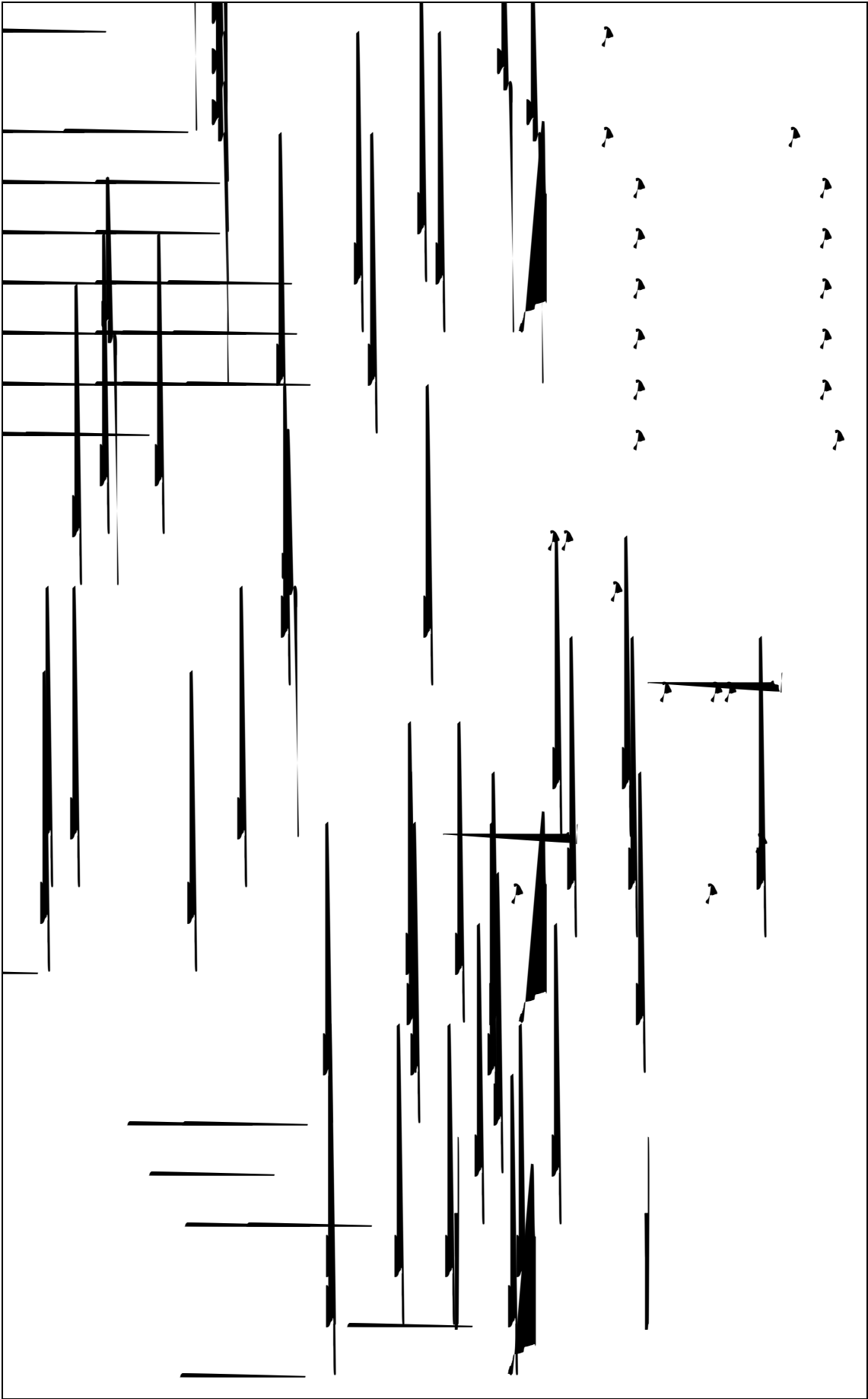
4.

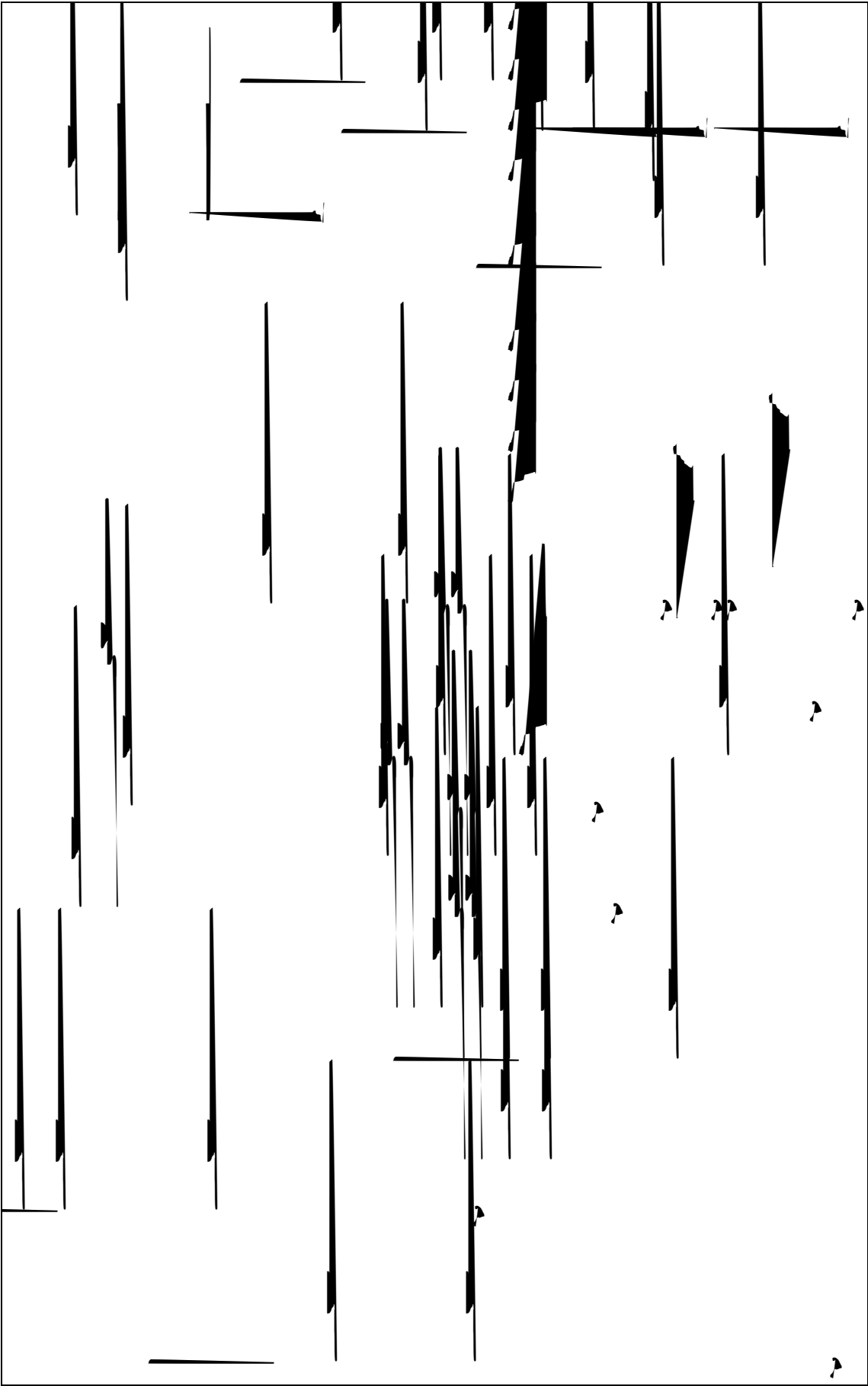
mapred 找共同朋友，数据格式如下

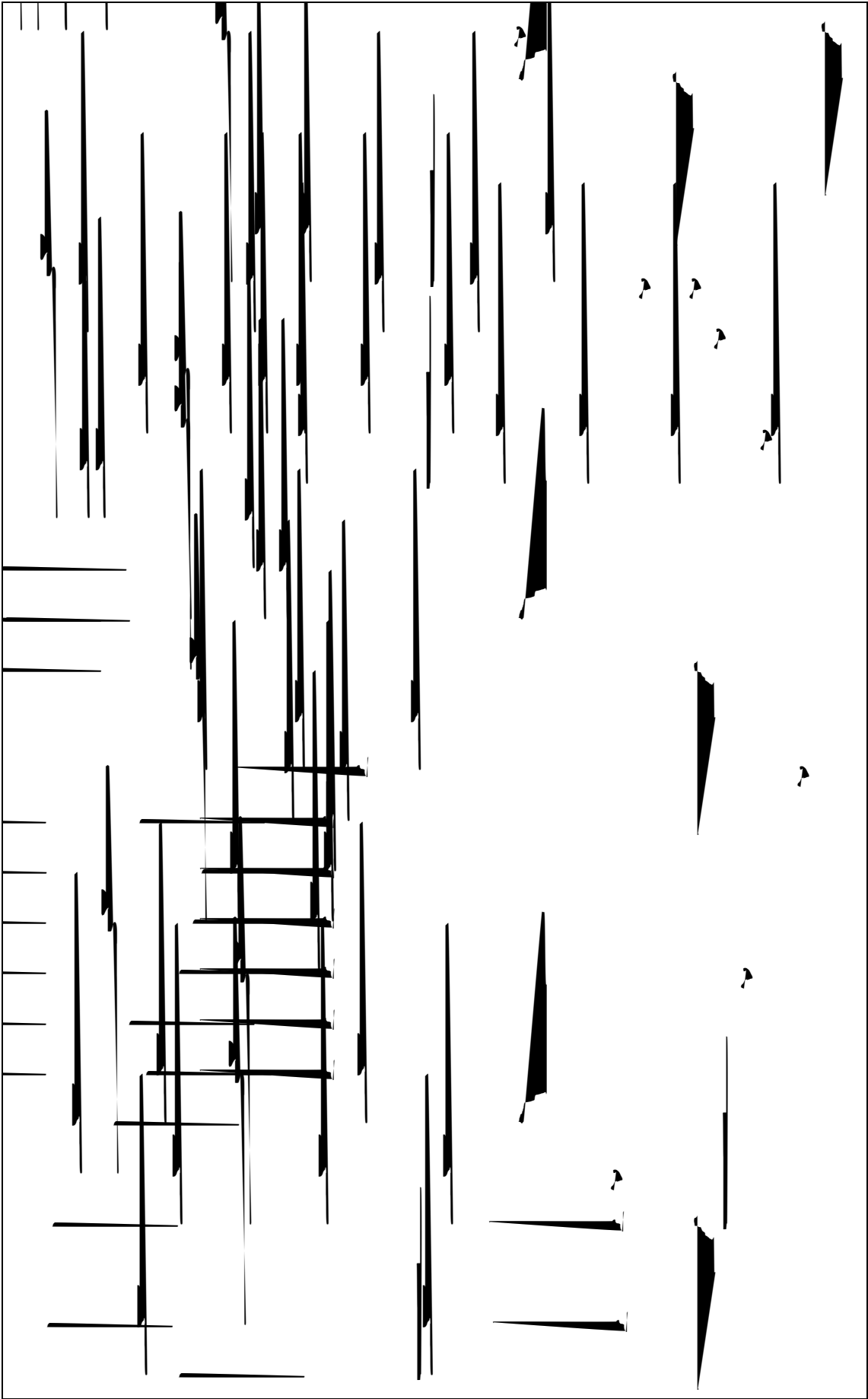
```
1. A B C D E F
2. B A C D E
3. C A B E
4. D A B E
5. E A B C D
6. F A
```

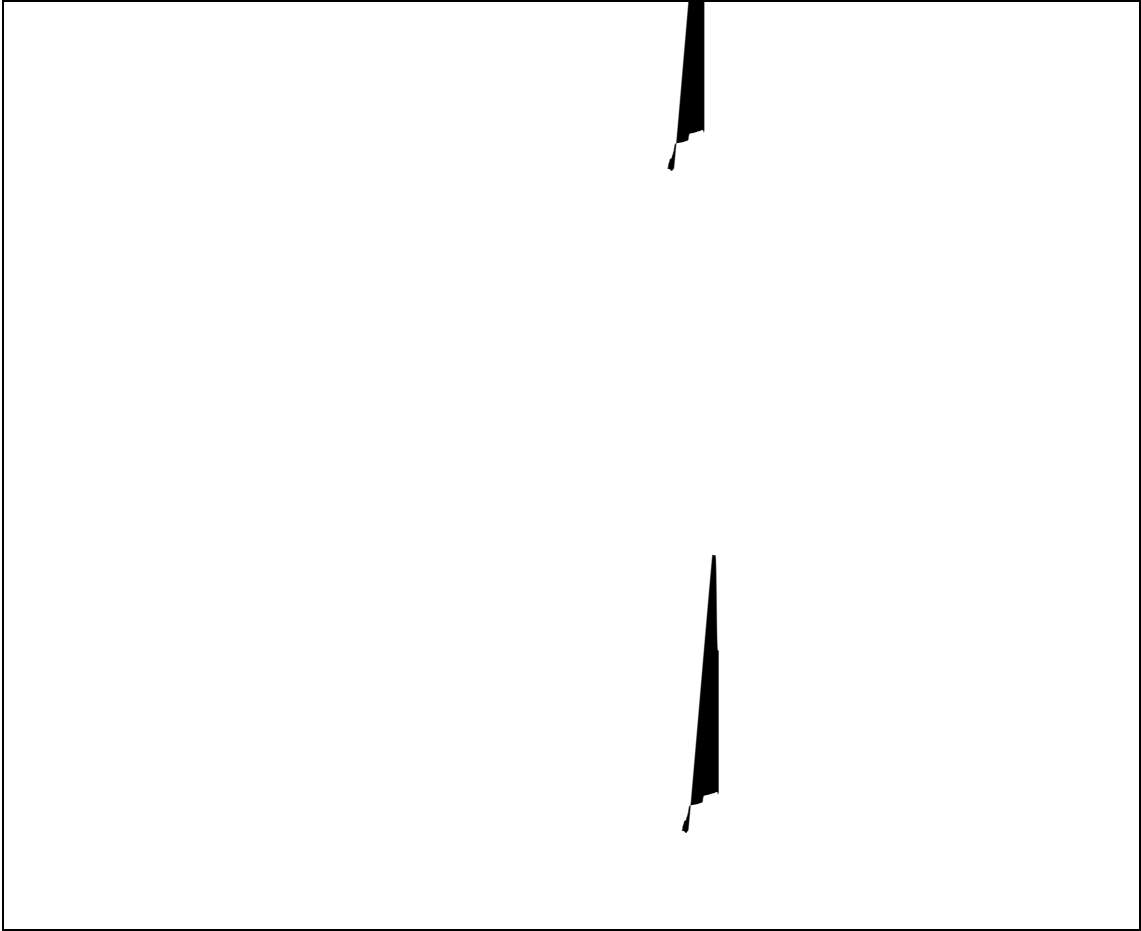
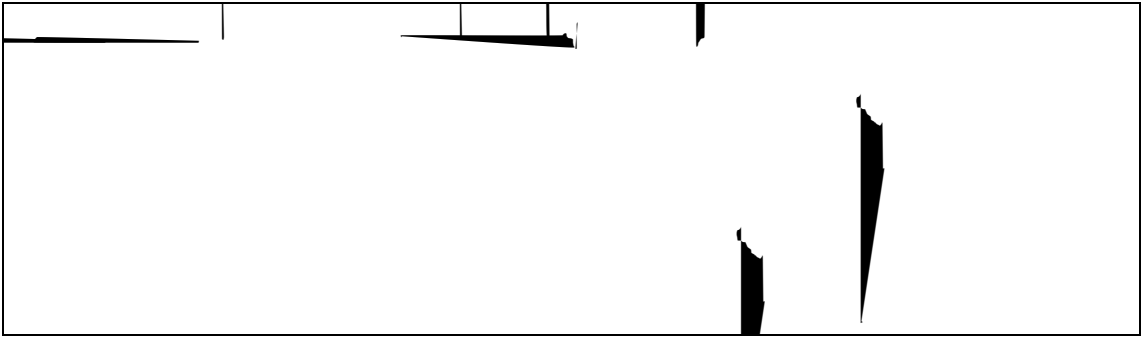
第一字母表示本人，其他是他的朋友，找出有共同朋友的人，和共同朋友是谁













## 5.

1 使用 Hive 或者自定义 MR 实现如下逻辑

product_no	lac_id	moment	start_time	user_id	county_id	staytime	city_id
13429100031	22554	8	2013-03-11 08:55:19.151754088	571	571	282	571
13429100082	22540	8	2013-03-11 08:58:20.152622488	571	571	270	571
13429100082	22691	8	2013-03-11 08:56:37.149593624	571	571	103	571
13429100087	22705	8	2013-03-11 08:56:51.139539816	571	571	220	571
13429100087	22540	8	2013-03-11 08:55:45.150276800	571	571	66	571
13429100082	22540	8	2013-03-11 08:55:38.140225200	571	571	133	571
13429100140	26642	9	2013-03-11 09:02:19.151754088	571	571	18	571
13429100082	22691	8	2013-03-11 08:57:32.151754088	571	571	287	571

字段解释：

**product\_no:** 用户手机号；

**lac\_id:** 用户所在基站；

**start\_time:** 用户在此基站的开始时间；

**staytime:** 用户在此基站的逗留时间。

根据 lac\_id 和 start\_time 知道用户当时的位置，根据 staytime 知道用户各个基站的逗留时长。根据轨迹合并连续基站的 staytime。

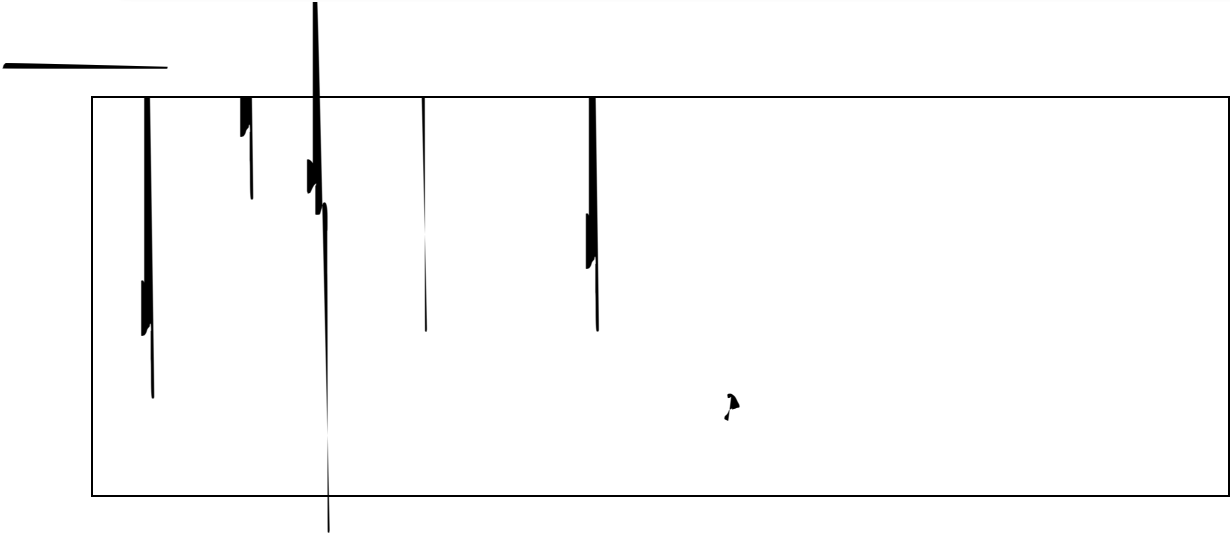
最终得到每一个用户按时间排序左邻一个基站逗留时长...

期望输出举例：

13429100082	22540	8	2013-03-11 08:58:20.152622488	571	571	270	571
13429100082	22691	8	2013-03-11 08:56:37.149593624	571	571	103	571
13429100082	22540	8	2013-03-11 08:55:38.140225200	571	571	133	571
13429100087	22705	8	2013-03-11 08:56:51.139539816	571	571	220	571
13429100087	22540	8	2013-03-11 08:55:45.150276800	571	571	66	571

## 6.

2.1 请随意使用各种类型的脚本语言实现：批量将指定目录下的所有文件中的\$HADOOP\_HOME\$替换成/home/ocetl/app/hadoop



## 7.

2.2 假设有 10 台主机，H1 到 H10，在开启 SSH 互信的情况下，编写一个或多个脚本实现在所有的远程主机上执行脚本的功能

例如：runRemoteCmd.sh "ls -l"

期望结果：

H1:

XXXXXXXX

XXXXXXXX

XXXXXXXX

H2:

XXXXXXXX

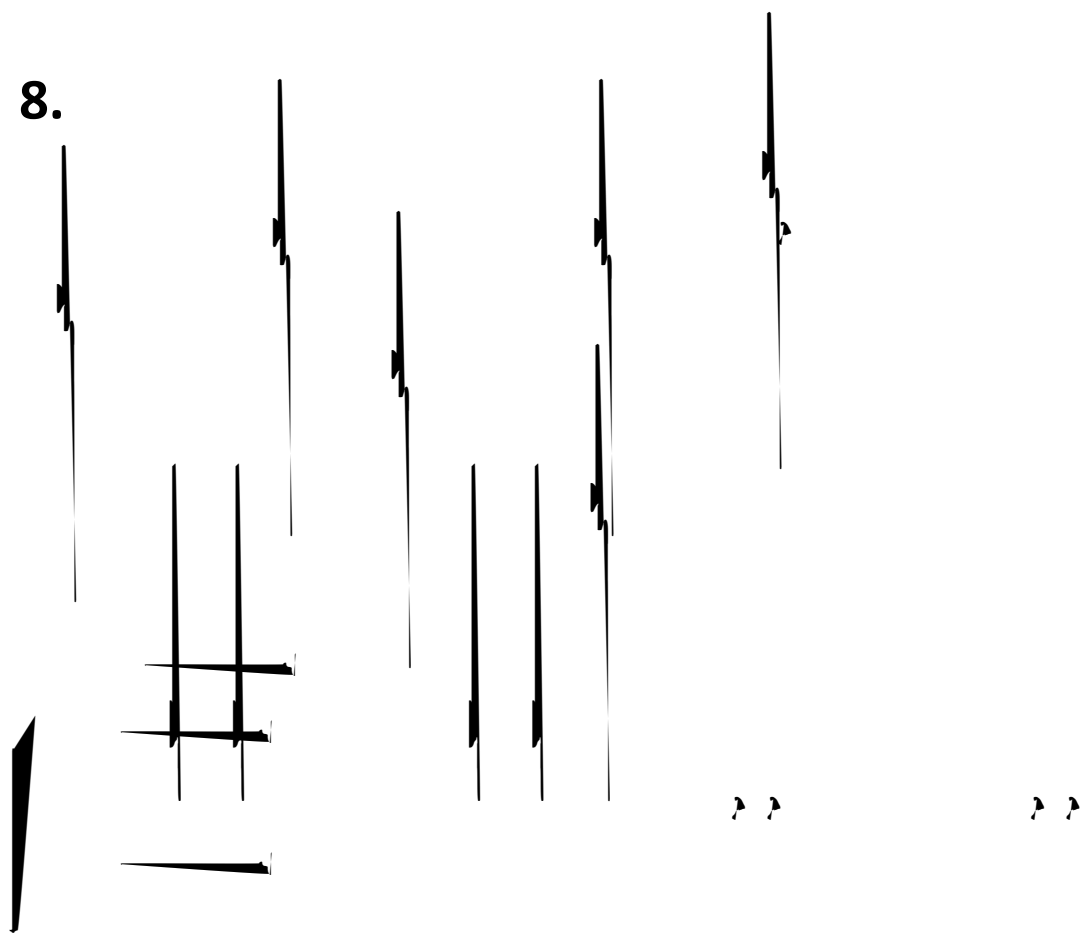
XXXXXXXX

XXXXXXXX

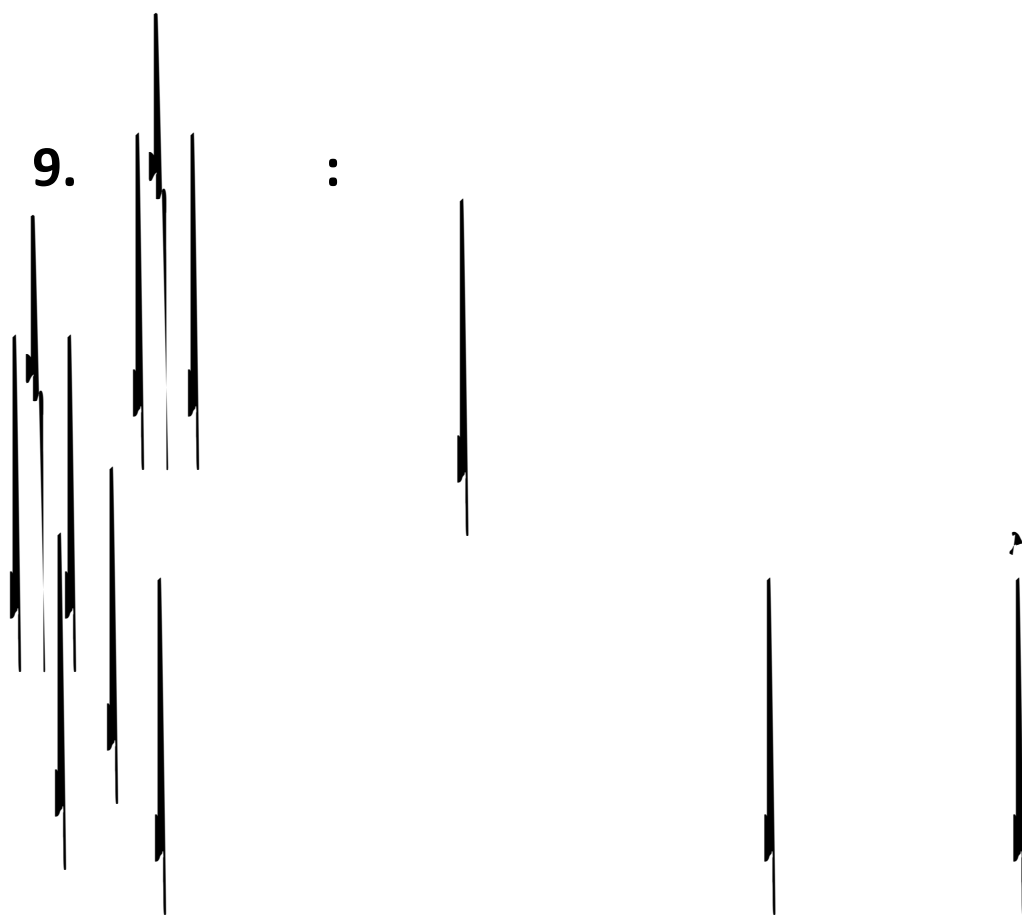
H3:

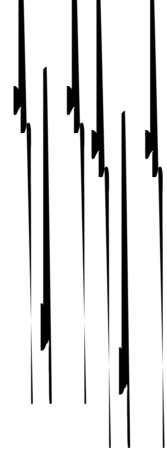
...

8.

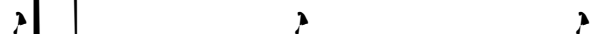
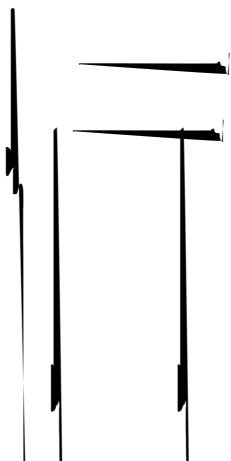


9.





10. :



**11.**

• •

[illegible]

1

1

2

2

7

**11.**

• •

[illegible]

1

1

2

2

7

12.

:

?

?

?

?

?

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

T

\_\_\_\_\_

?

\_\_\_\_\_

\_\_\_\_\_

?



2

13.

-





→

→

→

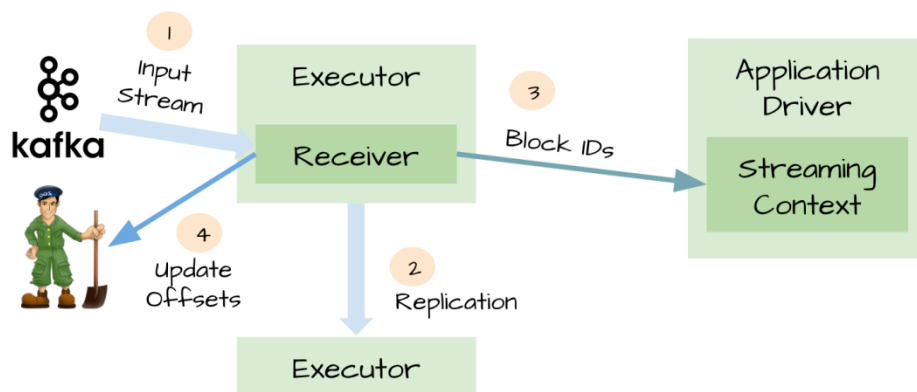
spark streaming 1.2

1. sources receivers
2. metadata driver checkpoint

### 3. WAL write ahead log

## 13.1.

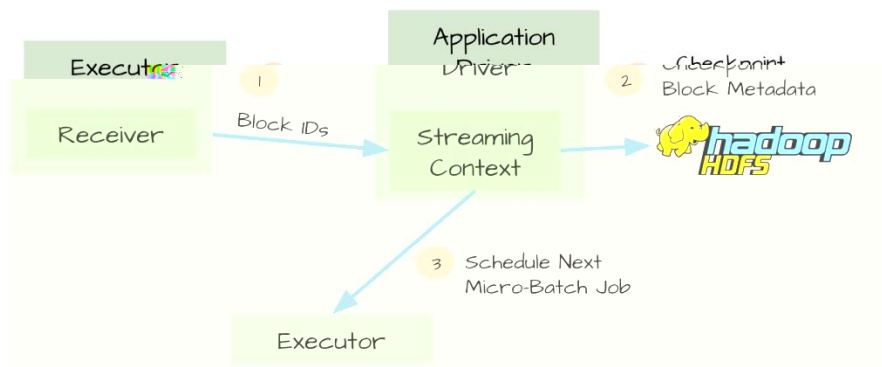
spark streaming receivers replication sources kafka  
 spark executors offsets zookeeper receivers  
 kafka crash receivers receiver



## 13.2.

S3 metadata sources receivers receivers checkpoint metadata driver  
 HDFS

- 
- 
- 



driver

metadata checkpoint

13.3.

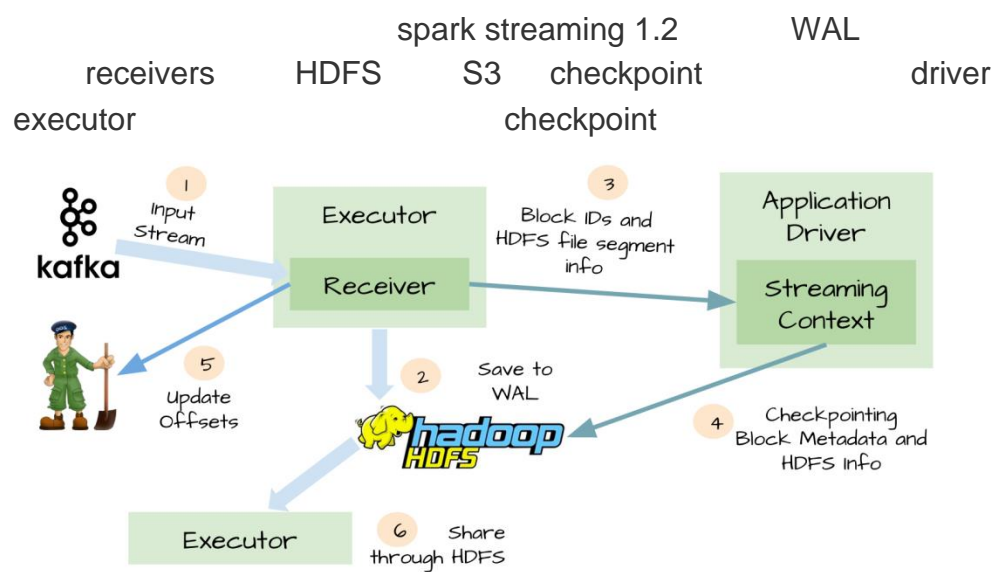
sources

receivers

metadata checkpoint

- 
- 
- 
- 
- 
- 
- 

13.4.



13.5.

-

-

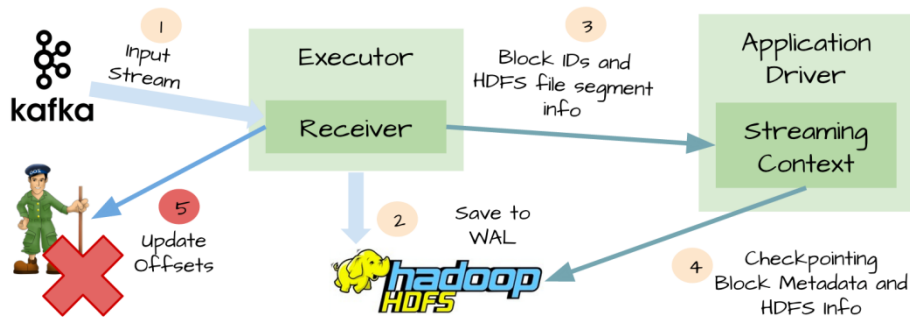
WAL

exactly-once

- Receivers

HDFS S3

- offset receivers



- Spark Streaming offset zookeeper Kafka
- receiver
- WAL kafka High-Level API zookeeper offsets

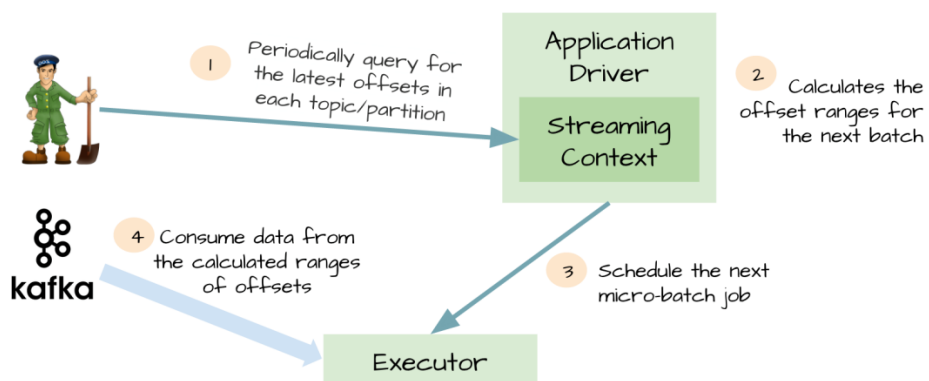
## 13.6.

WAL

- 
- 

## 13.7.

WAL exactly-once spark streaming1.3 Kafka direct  
API Spark driver batch offsets executor  
topics partitions Kafka



1. kafka receivers executor Kafka API

2. WAL

3. exactly-once

WAL

13.8.

spark streaming

exactly-once

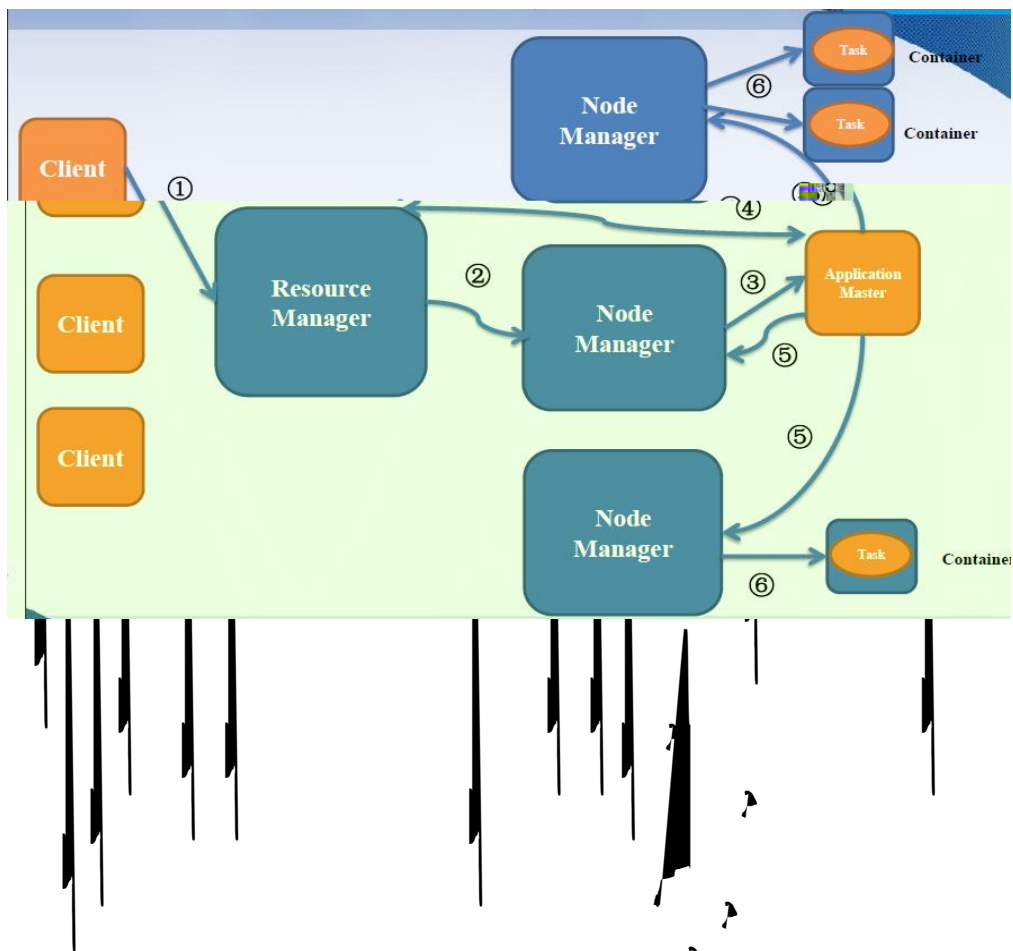
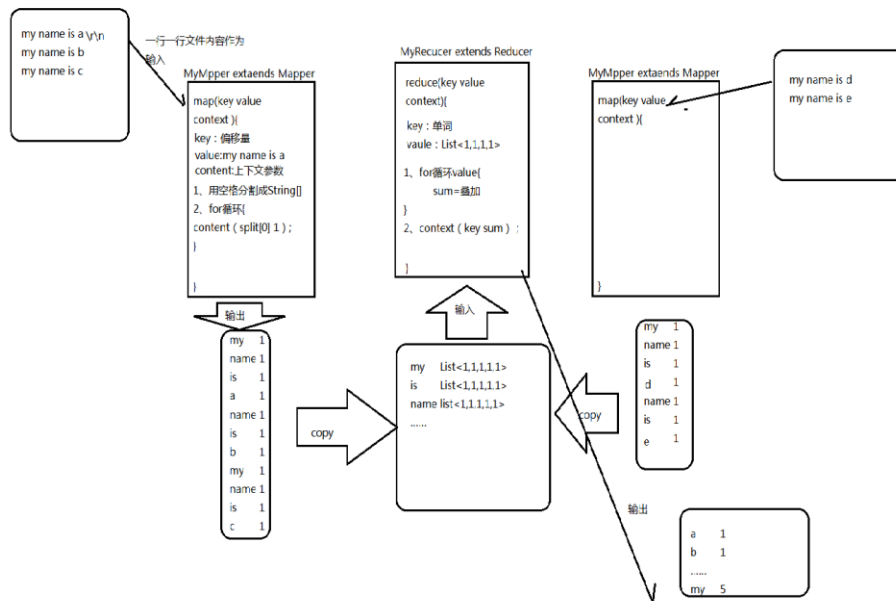
spark streaming

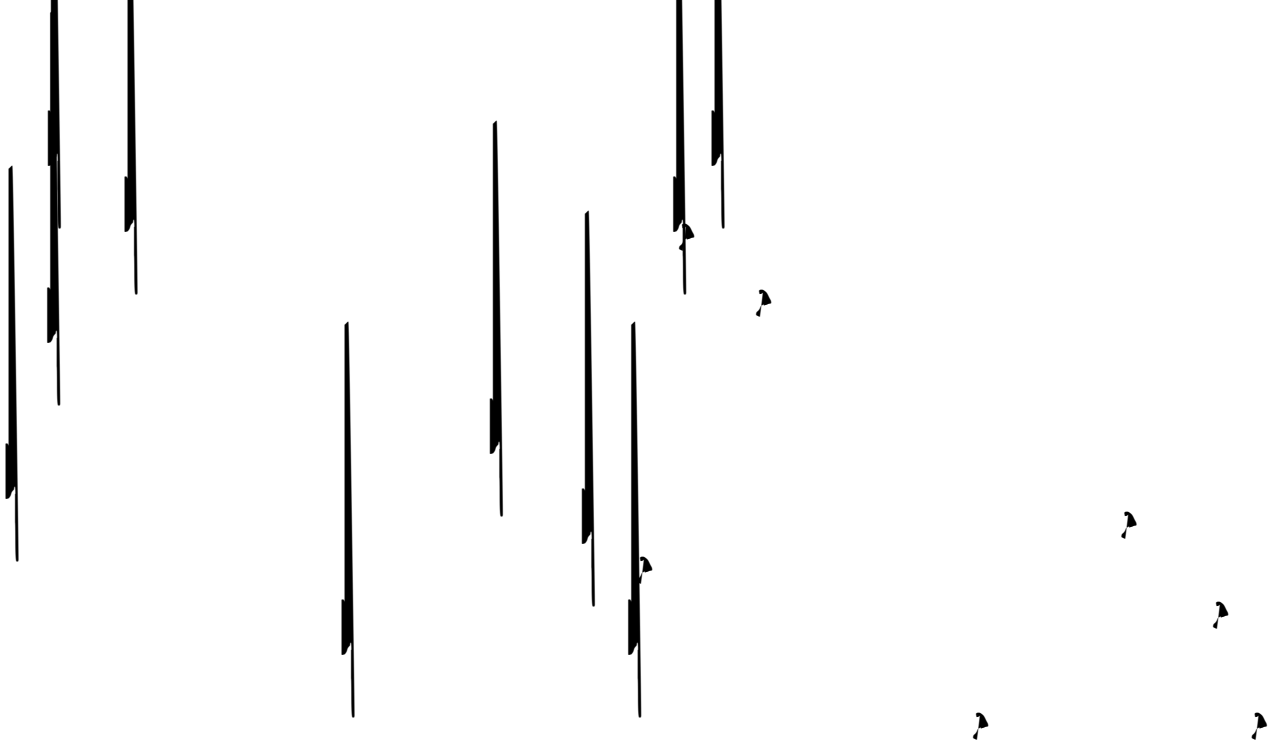


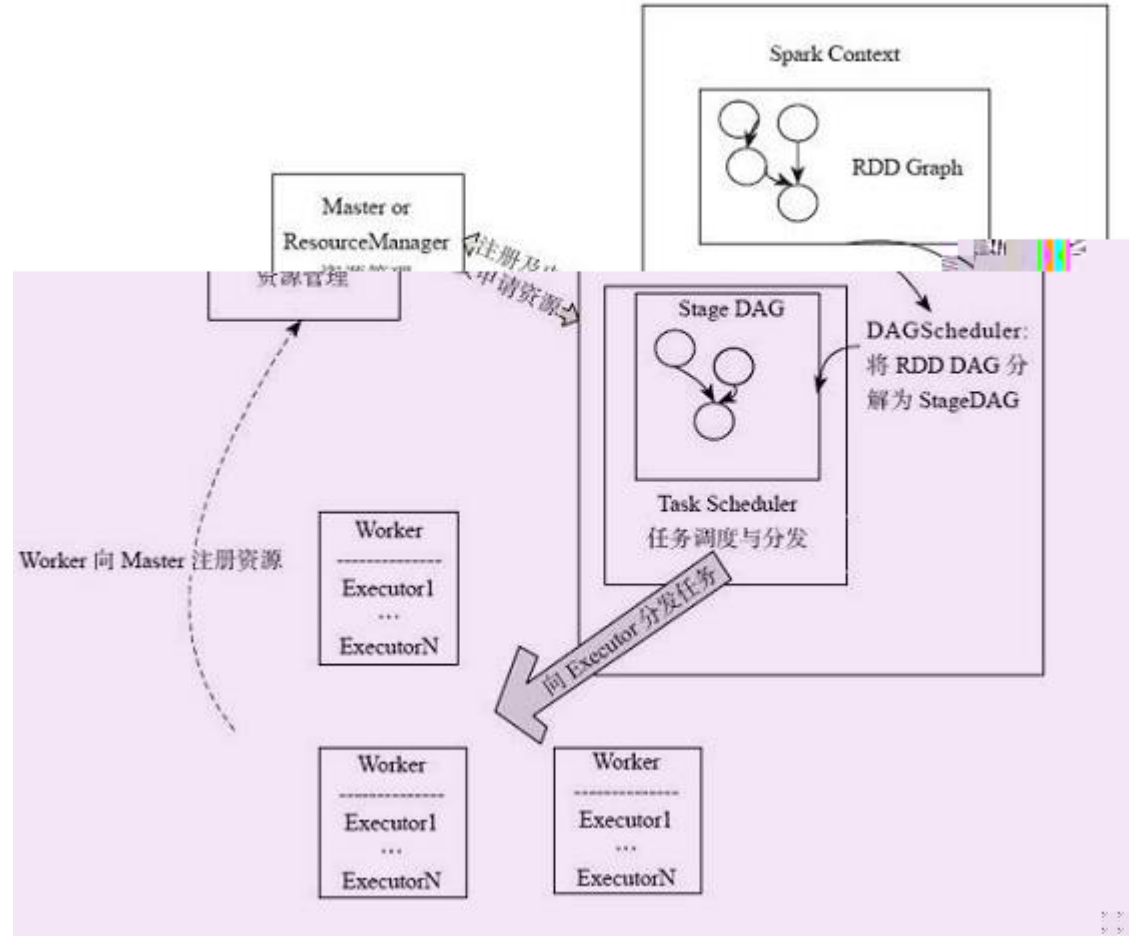






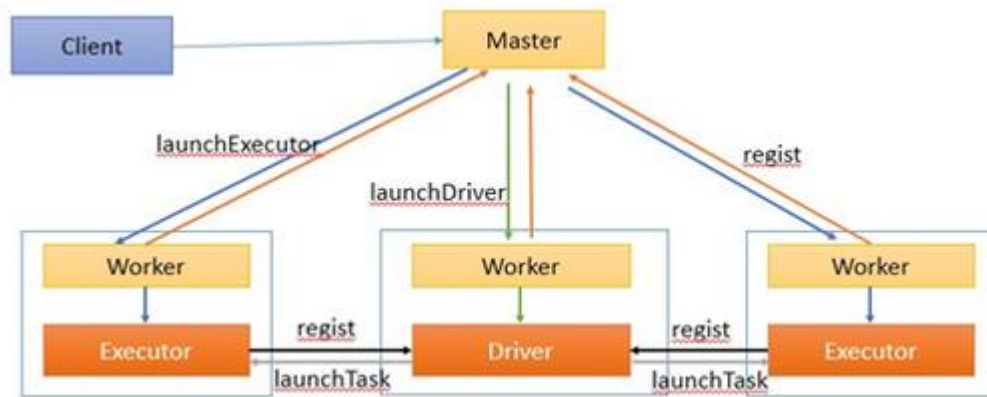






Driver

Worker



Driver

