云计算与大数据

基于 Hadoop 和 zookeeper 的心跳线实验

实验报告

1 实验要求

在 Hadoop MapReduce 的框架下（前 8 周搭建的环境中），利用 zookeeper 实现 slave 和 master 之间的心跳线维护。

具体做法： 以 map 为单位，在一群 map 中选举一个作为 master，其余作为 slave。master 指定 zookeeper 的目录作为心跳信息的接收路径，master 定期从 zookeeper 中检查是否收到 slave 的心跳信息。slave 定期向 master 的指定目录写入心跳信息。

2 实验环境

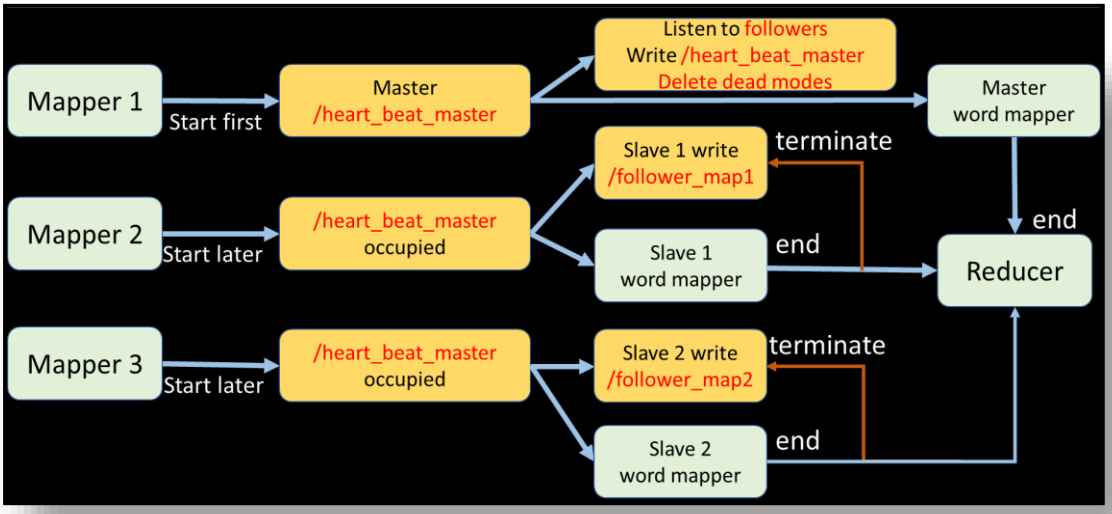
操作系统：CentOS 7

分布式框架：Hadoop 2.7.3

集群：3 台基于 VMWare Workstation 搭建的虚拟机 - hd-master, hd-slave1, hd-slave2，每台分配内存 4G。

其他环境：zookeeper 3.5.8；python 2.7.5

3 技术路线



3.1 zookeeper 节点及功能设计

ZooKeeper 是一个分布式的、开放源码的分布式应用程序协调服务，是 Google 的 Chubby 一个开源的实现，是 Hadoop 和 Hbase 的重要组件。它是一个为分布式应用提供一致性服务的软件，提供的功能包括：配置维护、域名服务、分布式同步、组服务等。ZooKeeper 支持大部分开发语言，除了某些特定的功能只支持 Java 和 C。python 通过 kazoo 可以实现操作 ZooKeeper。官方文档可见:

https://kazoo.readthedocs.io/en/latest/api/client.html。

3.1.1 master - /heart_beat_master

Zookeeper 内部的组织结构是树结构，父节点下面可以创建子节点，每个节点可以设置 value 值。利用这一特性，我设计了一个节点/heart_beat_master 用于记录 master 监听到的所有心跳。事实上，这一节点在最初始的时候会被 setup_zookeeper.py 创建并赋初值 b'this is a master map!'。然后，当某个 map 占用了这个节点，会用自己的名字（如 map1034677）给它赋值，相当于记录哪一个 map 现在扮演的是 master 的角色。之后，master 监听到的所有心跳信息都会被记录在这个节点里，如“Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive”。如果 master 发现某个 slave 经过 0.02 秒（自己设定的 timeout，可以修改）都没有发送心跳，判定该节点死亡，则也会把判定信息和随后的删除 slave 节点操作写入这个节点，如

“Tue Dec 29 19:18:39 2020: follower map1609240709.52 has died!

delete node map1609240709.52“

3.1.2 slave - /heart_beat

/heart_beat 是所有 slave 节点的父节点，当一个 slave map 启动，便会在这个节点下利用启动时间戳来创建与自己对应的子节点，并赋初值“this is a map!”。例如

/follower_map1609234444.71

/follower_map1609234453.49

之后只要 map 还在工作，就会每 0.005 秒向自己对应的节点写入带时间戳的心跳信息，如“alive at 1609240709.574485!”。当 map 完成工作，心跳发送便停止。而彼处的 master 判断该节点对应的 map 已经死亡时，会删除这个节点。

3.2 mapper 功能设计

如何起多个 map：由于我们的设定是在一个 task 下面起多个 map。所以我采用之前运行过的 word count 任务作为整体 task，通过在 input 文件夹下放入多个文件（实验中展示的是 3 个文件的情况）来达到起多个 map 的效果。

如何用 python 起 map：尽管 Hadoop 框架是用 java 写的，但是 Hadoop 程序不限于 java，可以用 python、C++、ruby 等。本实验中我直接用 python 写了一个 MapReduce 实例，而不是用 Jython 把 python 代码转化成 jar 文件。使用 python 写 MapReduce 的“诀窍”是利用 Hadoop 流的 API，通过 STDIN(标准输入)、STDOUT(标准输出)在 Map 函数和 Reduce 函数之间传递数据。我们唯一需要做的是利用 Python 的 sys.stdin 读取输入数据，并把我们的输出传送给 sys.stdout。Hadoop 流将会帮助我们处理别的任何事情。

心跳与任务的关系：而在 word count 中，mapper 的任务就是读取文件中的每一个单词，并输出(word, 1)的键值对。这一逻辑非常简单直接。但是如果要对 map 进行心跳监听，就意味着 map 在做上述任务的同时应当不断发送心跳，而做完以后立刻停止发送心跳，这样才能起到“监控某个 map 是死是活”的心跳线的作用。

并行处理：如何在 map 进行上述任务的同时写入心跳呢？我认为从逻辑上讲，这二者本身就是并行的，所以代码也应当写成并行的。于是我让主函数先启动“发送心跳”的子进程，于是当子进程不断发送心跳时，主函数依然会继续执行输出键值对的任务，当主函数执行完毕，便中止子进程，于是心跳发送停止。

同理，作为 master 的 map 进行自己任务的同时要对其他 map 进行监听，也应当是并行的。“监听心跳”相对于“输出键值对”也是子进程。

Master 选举：但接下来的问题是，写入心跳的方式确定了，监听心跳的 master 应该怎么选举呢？我认为作为一个监听者，理论上应当监听完全，最自然的想法就是“先到先得”，哪个 map 先运行，哪个 map 就成为 master 去监听其他 map 的心跳。于是我设计了/heart_beat_master 节点。对于一个 map，它应当先判断自己是 master 还是 slave，只要

/heart_beat_master 节点的值还是初值，就说明尚未有 map 进行监听，可以接管这一节点担当 master 的角色，开启监听子进程，但只要它的值不是初值，就说明已经有 map 在监听，它应当乖乖成为 slave，去开启发送心跳的子进程。

3.3 reducer 功能设计

由于我采用之前运行过的 word count 任务作为整体 task，所以对应的 reducer 和普通的 reducer 是一样的，主要功能便是整合输入的键值对，把相同的单词放在一起计数，最后输出每个单词的词频到指定目录下。

3.4 心跳日志的展示设计

由于上述心跳发送和监听过程都是通过修改或获取 zookeeper 里面节点的值来实现的，要显示地展现整个过程中每个节点值的变化，只能通过翻找每台机器上 zookeeper 的日志，这样太过于麻烦，于是我另外设计了两个进程 watch_master.py 和 watch_slave.py，专门用于全程 watch（监控）zookeeper 里面/heart_beat_master 和/heart_beat 两个目录下所有节点以及所有节点值的变化，每个节点对应一个输出文件，输出到指定目录下。这便于我直接观察和分析心跳发送和接收的全过程。

4 代码说明

4.1 mapper.py

首先导入需要的包库，kazoo 是用于连接 zookeeper 的包库，multiprocessing 是用于开启和管理多线程（进程）的包库。

mapper.py

```
#!/usr/bin/env python
#coding:utf-8
import sys
from kazoo.client import KazooClient
import time
import os
from multiprocessing import *

```

然后是主要的函数，第一个函数 heartbeat 是发送心跳的函数，传入参数为 map 启动的时间戳（也就是对应 zookeeper 节点的名字），主要工作是每隔 0.005 秒往对应节点内写入心跳信息。

```
def heartbeat(timestamp):
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    while True:
        time.sleep(0.005)
        time_now = time.time()
        zk.set('/heart_beat/map%s' %timestamp, b'alive at %f!' %time_now)
```

第二个函数 listener 是监听某个节点心跳的函数，传入参数为需要监听的节点名称，通过轮询方式监听对应节点的值，如果发现该节点超过 0.02 秒都没有心跳信息，则宣布该节点死亡，就删除对应节点并停止对该节点的监听（本进程结束）。

第三个函数 all_listen 是整体监听函数，事实上它只负责监控/heart_beat 下面是否有新的节点，如果有则开启一个 listener（上述第二个函数）去监听它。

接下来是主函数的 setup 阶段，主要目的就是判断自己是否能成为 master，如果能，则开启监听子进程 all_listen，如果不能则开启心跳发送子进程 heartbeat。

```python
def listener(node):
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    old_value = 'start'
    while True:
        time.sleep(0.02)
        value = zk.get('/heart_beat/%s' %node)
        if value == old_value: # the value did not change for 0.02 seconds we will consider it died
            message = time.asctime( time.localtime(time.time()) ) + ": follower "+ node+ " has died!\n"
            zk.set('/heart_beat_master', message.encode(encoding='UTF-8',errors='strict'))
            message = "delete node %s" %node
            zk.set('/heart_beat_master', message.encode(encoding='UTF-8',errors='strict'))
            zk.delete('/heart_beat/%s' %node,recursive=False)
            zk.stop()
            break
        else:
            message = time.asctime( time.localtime(time.time()) )+ ": follower %s still alive\n" %node
            zk.set('/heart_beat_master', message.encode(encoding='UTF-8',errors='strict'))
            old_value = value
```

```python
def all_listen(timestamp):
    import logging
    logging.basicConfig()
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    if zk.connected:
        zk.set('/heart_beat_master', b"zookeeper: CONNECTED\nmaster map%s\n"%timestamp)
    old_nodes = []
    while True:
        nodes = zk.get_children('/heart_beat')
        if nodes == old_nodes: # no new nodes
            continue
        else:
            for node in nodes:
                if node in old_nodes:
                    continue
                else: # this one is new node
                    # start a listener for heatbeat
                    p = Process(target = listener, args = [node])
                    p.start()
            old_nodes = nodes
```

```python
if __name__ == "__main__":
    import logging
    logging.basicConfig()
    timestamp = str(time.time())
    # 判断自己能否成为master
    # connect zookeeper
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    value = zk.get('/heart_beat_master')
    if value[0] == b'this is a master map!': # 尚未有监听进程
        # start listener
        p = Process(target = all_listen, args = [timestamp])
        p.start()
        zk.stop()
    else:
        # 创建节点：makepath 设置为 True ，父节点不存在则创建，其他参数不填均为默认
        zk.create('/heart_beat/map%s' %timestamp,b'this is a map!',makepath=True)
        # 操作完后，别忘了关闭zk连接
        zk.stop()
        # start heatbeat
        p = Process(target = heartbeat, args = [timestamp])
        p.start()
```

4.2 reducer.py

正常的 reducer 代码，主要功能就是汇集输入的键值对，输出每个单词以及词频。

```python
#!/usr/bin/env python
#coding:utf-8
from operator import itemgetter
import sys

current_word = None
current_count = 0
word = None

for line in sys.stdin:
    line = line.strip()
    word, count = line.split('\t', 1)
    try:
        count = int(count)
    except ValueError:  #count如果不是数字的话，直接忽略掉
        continue
    if current_word == word:
        current_count += count
    else:
        if current_word:
            print "%s\t%s" % (current_word, current_count)
        current_count = count
        current_word = word

if word == current_word:  #不要忘记最后的输出
    print "%s\t%s" % (current_word, current_count)
```

注：其他代码见附录

5 实验结果截图

5.1 准备工作

开启三台虚拟机，分别打开终端，分别登入 Root 用户。

清理 HADOOP 日志

rm -rf /opt/linuxsir/hadoop/logs/*.*

ssh root@192.168.31.130 rm -rf /opt/linuxsir/hadoop/logs/*.*

ssh root@192.168.31.131 rm -rf /opt/linuxsir/hadoop/logs/*.*

启动 HADOOP

cd /opt/linuxsir/hadoop/sbin

./start-dfs.sh

./start-yarn.sh

然后在**每一台机子**上开启 zookeeper

source /etc/profile

zkServer.sh start

5.2 初始化

清理之前所有的 python 进程以防干扰：

cd /opt/linuxsir/hadoop

pkill -9 python

系统初始化，主要是看/heart_beat_master 节点是否存在，如果不存在则创建，并赋初值 b'this is a master map!'：

python setup_zookeeper.py

开启日志输出进程，主要是为了在 Hadoop 根目录下就简洁方便地看到 zookeeper 各

节点的变化：

nohup python -u watch_slave.py > watch_slave.out 2>&1 &

nohup python -u watch_master.py > heart_beat_master.txt 2>&1 &

5.3 mapreduce 实验主体

通过下列代码可以让 Hadoop 顺利运行我们之前写的 mapper.py 和 reducer.py。前三行主要是清除之前可能残留的结果文件，以免 Hadoop 报错说文件已存在。最后一行通过 stream 流指定 mapper 和 reducer，指定输入目录和输出目录，便可直接运行。Word count 的结果文件在/test_out 目录下。

cd /opt/linuxsir/hadoop

bin/hdfs dfs -rm /test_out/*

bin/hdfs dfs -rmdir /test_out

bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -input /hdfs_in/datas/* -output /test_out -file ./mapper.py -mapper "python mapper.py" -file ./reducer.py -reducer "python reducer.py"

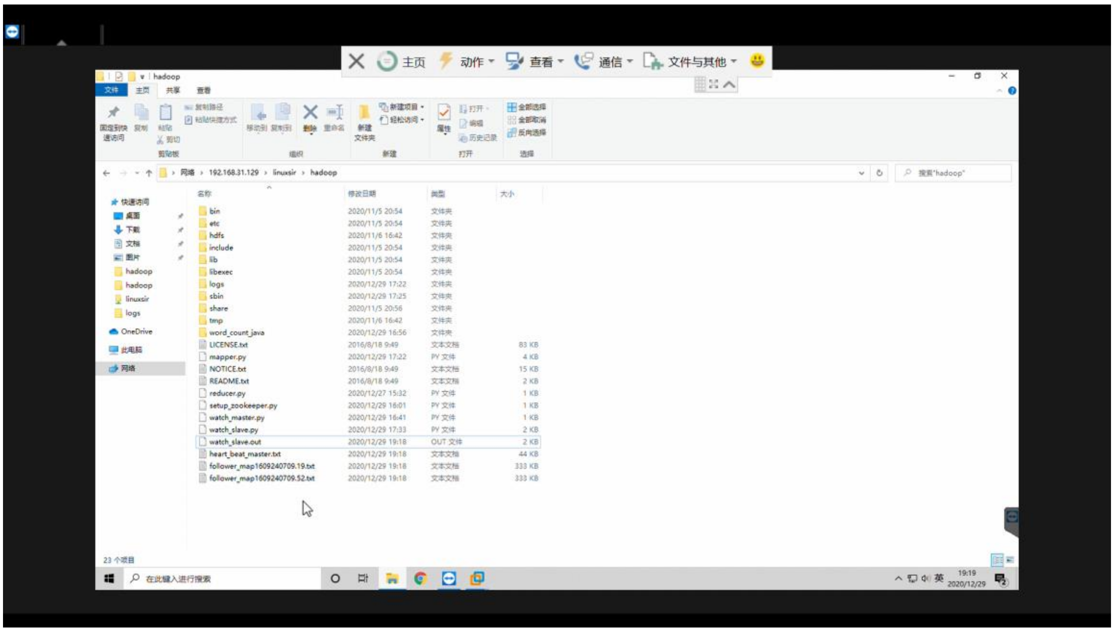
结果如图所示：

```
[root@hd-master hadoop]# cd /opt/linuxsir/hadoop
[root@hd-master hadoop]# bin/hdfs dfs -rm /test_out/*
20/12/29 19:18:06 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /test_out/_SUCCESS
20/12/29 19:18:06 INFO fs.TrashPolicyDefault: Namenode trash configuration: Deletion interval = 0 minutes, Emptier interval = 0 minutes.
Deleted /test_out/part-00000
[root@hd-master hadoop]# bin/hdfs dfs -rmdir /test_out
[root@hd-master hadoop]# bin/hadoop jar share/hadoop/tools/lib/hadoop-streaming-2.7.3.jar -input /hdfs_in/datas/* -output /test_out -file ./mapper.py -mapper "python mapper.py" -file ./reducer.py -reducer "python reducer.py"
20/12/29 19:18:10 WARN streaming.StreamJob: -file option is deprecated, please use generic option -files instead.
packageJobJar: [./mapper.py, ./reducer.py, /tmp/hadoop-unjar3444951165087766737/] [] /tmp/streamjob5827141908221117963.jar tmpDir=null
20/12/29 19:18:11 INFO client.RMProxy: Connecting to ResourceManager at hd-master/192.168.31.129:9032
20/12/29 19:18:11 INFO client.RMProxy: Connecting to ResourceManager at hd-master/192.168.31.129:9032
20/12/29 19:18:13 INFO mapred.FileInputFormat: Total input paths to process : 3
20/12/29 19:18:13 INFO mapreduce.JobSubmitter: number of splits:3
20/12/29 19:18:13 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1609240638654_0001
20/12/29 19:18:14 INFO impl.YarnClientImpl: Submitted application application_1609240638654_0001
20/12/29 19:18:14 INFO mapreduce.Job: The url to track the job: http://hd-master:9099/proxy/application_1609240638654_0001/
20/12/29 19:18:14 INFO mapreduce.Job: Running job: job_1609240638654_0001
20/12/29 19:18:22 INFO mapreduce.Job: Job job_1609240638654_0001 running in uber mode : false
20/12/29 19:18:22 INFO mapreduce.Job:  map 0% reduce 0%
20/12/29 19:18:32 INFO mapreduce.Job:  map 22% reduce 0%
20/12/29 19:18:35 INFO mapreduce.Job:  map 37% reduce 0%
20/12/29 19:18:36 INFO mapreduce.Job:  map 50% reduce 0%
20/12/29 19:18:39 INFO mapreduce.Job:  map 64% reduce 0%
20/12/29 19:18:41 INFO mapreduce.Job:  map 100% reduce 0%
```

```
        Map-Reduce Framework
                Map input records=77672
                Map output records=629206
                Map output bytes=4817895
                Map output materialized bytes=6076325
                Input split bytes=295
                Combine input records=0
                Combine output records=0
                Reduce input groups=82342
                Reduce shuffle bytes=6076325
                Reduce input records=629206
                Reduce output records=82342
                Spilled Records=1258412
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=513
                CPU time spent (ms)=13760
                Physical memory (bytes) snapshot=877473792
                Virtual memory (bytes) snapshot=8657264640
                Total committed heap usage (bytes)=810024960
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=3671131
        File Output Format Counters
                Bytes Written=880874
20/12/29 19:18:50 INFO streaming.StreamJob: Output directory: /test_out
```

```
[root@hd-master hadoop]# bin/hdfs dfs -ls /test_out
Found 2 items
-rw-r--r--   2 root supergroup          0 2020-12-29 19:18 /test_out/_SUCCESS
-rw-r--r--   2 root supergroup     880874 2020-12-29 19:18 /test_out/part-00000
[root@hd-master hadoop]# bin/hdfs dfs -cat /test_out/part-00000 | head -n 10
"(Lo)cra"       1
"1490   1
"1498," 1
"35"    1
"40,"   1
"A      2
"AS-IS".        1
"A_     1
"Absoluti       1
"Alack! 1
```

然后在 Hadoop 根目录下可以看到多了几个 txt 文件，其中 heart_beat_master.txt 记录的就是监听到的信息，follower 开头的两个 txt 文件记录的是两个 slave 的信息：



打开这几个文件分别看一眼，可以看到预期的心跳信息和操作都已发生：

```
heart_beat_master.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
this is a master map!
zookeeper: CONNECTED
master map1609240705.83

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:29 2020: follower map1609240709.19 still alive
```

heart_beat_master.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 still alive

Tue Dec 29 19:18:39 2020: follower map1609240709.52 has died!

delete node map1609240709.52
```

follower_map1609240709.19.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
this is follower node map1609240709.19
zookeeper: CONNECTED
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.285277!', ZnodeStat(czxid=8589969069,
mzxid=8589969074, ctime=1609240709245, mtime=1609240709286, version=1, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.305148!', ZnodeStat(czxid=8589969069,
mzxid=8589969075, ctime=1609240709245, mtime=1609240709306, version=2, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.313927!', ZnodeStat(czxid=8589969069,
mzxid=8589969077, ctime=1609240709245, mtime=1609240709314, version=3, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.334425!', ZnodeStat(czxid=8589969069,
mzxid=8589969078, ctime=1609240709245, mtime=1609240709334, version=4, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.342176!', ZnodeStat(czxid=8589969069,
mzxid=8589969080, ctime=1609240709245, mtime=1609240709342, version=5, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.349330!', ZnodeStat(czxid=8589969069,
mzxid=8589969081, ctime=1609240709245, mtime=1609240709349, version=6, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.357326!', ZnodeStat(czxid=8589969069,
mzxid=8589969082, ctime=1609240709245, mtime=1609240709357, version=7, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.390155!', ZnodeStat(czxid=8589969069,
mzxid=8589969084, ctime=1609240709245, mtime=1609240709400, version=8, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.440464!', ZnodeStat(czxid=8589969069,
mzxid=8589969086, ctime=1609240709245, mtime=1609240709452, version=9, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.464041!', ZnodeStat(czxid=8589969069,
mzxid=8589969087, ctime=1609240709245, mtime=1609240709464, version=10, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969069))!
set value of node /heart_beat/follower_map1609240709.19 as ('alive at 1609240709.477418!', ZnodeStat(czxid=8589969069,
mzxid=8589969089, ctime=1609240709245, mtime=1609240709477, version=11, cversion=0, aversion=0, ephemeralOwner=0,
```

follower_map1609240709.52.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
this is follower node map1609240709.52
zookeeper: CONNECTED
set value of node /heart_beat/follower_map1609240709.52 as ('this is a map!', ZnodeStat(czxid=8589969098, mzxid=8589969098,
ctime=1609240709545, mtime=1609240709545, version=0, cversion=0, aversion=0, ephemeralOwner=0, dataLength=14, numChildren=0,
pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.574485!', ZnodeStat(czxid=8589969098,
mzxid=8589969108, ctime=1609240709545, mtime=1609240709575, version=1, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.582623!', ZnodeStat(czxid=8589969098,
mzxid=8589969110, ctime=1609240709545, mtime=1609240709583, version=2, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.590659!', ZnodeStat(czxid=8589969098,
mzxid=8589969112, ctime=1609240709545, mtime=1609240709591, version=3, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.598851!', ZnodeStat(czxid=8589969098,
mzxid=8589969116, ctime=1609240709545, mtime=1609240709599, version=4, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.607569!', ZnodeStat(czxid=8589969098,
mzxid=8589969118, ctime=1609240709545, mtime=1609240709608, version=5, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.615604!', ZnodeStat(czxid=8589969098,
mzxid=8589969122, ctime=1609240709545, mtime=1609240709616, version=6, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.623463!', ZnodeStat(czxid=8589969098,
mzxid=8589969124, ctime=1609240709545, mtime=1609240709623, version=7, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.630267!', ZnodeStat(czxid=8589969098,
mzxid=8589969126, ctime=1609240709545, mtime=1609240709630, version=8, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.636950!', ZnodeStat(czxid=8589969098,
mzxid=8589969129, ctime=1609240709545, mtime=1609240709637, version=9, cversion=0, aversion=0, ephemeralOwner=0,
dataLength=27, numChildren=0, pzxid=8589969098))!
set value of node /heart_beat/follower_map1609240709.52 as ('alive at 1609240709.647725!', ZnodeStat(czxid=8589969098,
mzxid=8589969131, ctime=1609240709545, mtime=1609240709649, version=10, cversion=0, aversion=0, ephemeralOwner=0,
```

6 碰到的问题及解决办法

6.1 利用 python 运行 mapreduce 任务

参考如下网址：

https://www.cnblogs.com/kaituorensheng/p/3826114.html

进行代码编写和运行，但是 Hadoop 一直报错但代码本地测试时并没有问题，不知道是哪里错，就很迷惑，并且 Hadoop 跑之前 java 写的的 word count 没问题，后来发现要把 mapper.py 和 reducer.py 放在 hadoop 的根目录下面。

6.2 环境配置

要在三台机子上安装 python 包 multiprocessing，报错说找不到 gcc，意思是没有 c++ 编译器，于是先重新挂载镜像光盘，然后 yum install gcc，再 pip 便可成功。

6.3 python 连接 zookeeper

第一次运行时报错"No handlers could be found for logger "kazoo.client"，经查阅资料，发现在初始化之前配置 log 即可：

import logging

logging.basicConfig()

6.4 时间设定

一开始设定 Watch_slave.py 停顿和 master 监听的 timeout 是一样的时间，看到的东西却比 master 少很多，看来 Watch_slave.py 本身写入磁盘确实很慢，因此我把 mapper.py 里面输出键值对的过程刻意延长了一些，然后 Watch_slave.py 并不采取隔一段时间看一眼的方式，而是 watch（有值的变化就输出）的方式，如此才真正达到"日志输出"的目的。

7 附录

7.1 setup_zookeeper.py

```python
from kazoo.client import KazooClient
import time

if __name__ == "__main__":
    import logging
    logging.basicConfig()
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    if zk.connected:
        print "connected\n"
    if zk.exists('/heart_beat_master'):
        zk.set('/heart_beat_master' ,b'this is a master map!')
    else:
        zk.create('/heart_beat_master' ,b'this is a master map!',makepath=True)
    zk.stop()
```

7.2 watch_master.py

```python
from kazoo.client import KazooClient
import time
from multiprocessing import *

if __name__ == "__main__":
    import logging
    logging.basicConfig()
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    if zk.connected:
        print "zookeeper: CONNECTED"
    else:
        print "zookeeper: UNCONNECTED"
    old_value = 'start'
    while True:
        value = zk.get('/heart_beat_master')
        if value != old_value:
            print value[0]
            old_value = value
```

7.3 watch_slave.py

```python
from kazoo.client import KazooClient
import time
from multiprocessing import *


def listener(node):
    f = open("follower_%s.txt" %node, 'w')
    f.write("this is follower node %s\n" %node)
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    if zk.connected:
        f.write("zookeeper: CONNECTED\n")
    else:
        f.write("zookeeper: UNCONNECTED\n")
    old_value = 'start'
    while True:
        value = zk.get('/heart_beat/%s' %node)
        if value == old_value:
            continue
        else:
            f.write("set value of node /heart_beat/follower_%s as %s!\n" %(node, value))
            old_value = value
```

```python
if __name__ == "__main__":
    import logging
    logging.basicConfig()
    zk = KazooClient(hosts='192.168.31.130:2181')
    zk.start()
    old_nodes = []
    while True:
        nodes = zk.get_children('/heart_beat')
        if nodes == old_nodes: # no new nodes
            continue
        else:
            for node in nodes:
                if node in old_nodes:
                    continue
                else: # this one is new node
                    # start a listener for heatbeat
                    p = Process(target = listener, args = [node])
                    p.start()
            old_nodes = nodes
```