

# Homework 4: Pagerank on Spark

## 一、Spark环境搭建

参照“how to deploy hadoop on netease cloud”中的博客链接，以及网上的其他资料，在服务器上搭建可运行spark程序的环境。

### 1.Scala安装

Spark的底层使用Scala语言开发，Scala语言对Spark的兼容性好，且实现简洁。此次作业中PageRank算法也通过Scala语言来实现。

首先下载Scala 2.11.7版本

```
wget https://downloads.lightbend.com/scala/2.11.7/scala-2.11.7.tgz
```

然后解压文件

```
tar -zxvf scala-2.11.7.tgz
```

删除源文件

```
rm -rf scala-2.11.7.tgz
```

在 `/etc/profile` 中配置相关环境变量，如下

```
## /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi

export SCALA_HOME=/usr/scala-2.11.7
export PATH=$PATH:$SCALA_HOME/bin
```

```
source /etc/profile
```

随后检查是否成功安装

```
scala -version
```

```
root@hadoop1:~# scala -version
Scala code runner version 2.11.7 -- Copyright 2002-2013, LAMP/EPFL
```

说明成功安装Scala 2.11.7版本

通过将文件夹也传到其他服务器上，进行相同的配置。

```
scp -r /usr/scala-2.11.7/ hadoop2:/usr/scala-2.11.7/
```

## 2.Spark安装

Spark是处理大规模数据的快速通用计算引擎，比起hadoop拥有更加优越的工作性能。

首先下载spark，我们采用2.2.0版本：

```
wget https://d3kbcqa49mib13.cloudfront.net/spark-2.2.0-bin-hadoop2.7.tgz
```

解压文件并修改文件夹名字

```
tar -zxvf spark-2.2.0-bin-hadoop2.7.tgz
rm -rf spark-2.2.0-bin-hadoop2.7.tgz
mv spark-2.2.0-bin-hadoop2.7 spark-2.2.0
```

随后修改 /etc/profile，最后加入以下两行：

```
export SPARK_HOME=/usr/local/spark-2.2.0
export PATH=$PATH:$SPARK_HOME/bin
```

接下来配置spark环境，首先把缓存的文 spark-env.sh.template 改为 spark-env.sh

```
cp conf/spark-env.sh.template conf /spark-env.sh
```

然后修改该文件，如下图

```
export JAVA_HOME=/usr/lib/jvm/default-java
export SCALA_HOME=/usr/scala-2.11.7
export HADOOP_HOME=/usr/local/hadoop
#export HADOOP_CONF_DIR=/usr/local/hadoop-2.7.2/etc/hadoop
export SPARK_MASTER_IP=hadoop1
export SPARK_WORKER_MEMORY=4g
export SPARK_WORKER_CORES=2
export SPARK_WORKER_INSTANCES=1
```

再修改slaves文件，添加各个工作节点的名称

```
hadoop2
hadoop3
hadoop4
hadoop5
```

同步其他结点上的spark配置

```
scp -r /usr/local/spark-2.2.0/ hadoop5:/usr/local/spark-2.2.0/
```

启动HDFS文件系统和Spark，验证是否安装成功

```
start-dfs.sh
./sbin/start-all.sh
jps
```

在主机上可以看到以下进程：


```
root@hadoop1:/usr/local/spark-2.2.0/sbin# jps
2625 Master
2100 NameNode
2697 Jps
2457 SecondaryNameNode
```

在hadoop2上可以看到以下进程

```
root@hadoop2:~# jps
1201 DataNode
1441 Jps
1359 Worker
```

说明已正常启动了spark

打开Spark的Web UI界面，即访问 hadoop1:8080，可见到所有结点的运行情况

 **Spark Master at spark://hadoop1:7077**

URL: spark://hadoop1:7077  
REST URL: spark://hadoop1:8086 (cluster mode)  
Alive Workers: 4  
Cores in use: 0 Total, 0 Used  
Memory in use: 16.0 GB Total, 0.0 B Used  
Applications: 0 Running, 0 Completed  
Drivers: 0 Running, 0 Completed  
Status: ALIVE

**Workers**

Worker Id	Address	State	Cores	Memory
worker-20191119145104-10.173.32.10-38000	10.173.32.10-38000	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191119145104-10.173.32.11-32840	10.173.32.11-32840	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191119145104-10.173.32.12-40669	10.173.32.12-40669	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191119145104-10.173.32.9-40364	10.173.32.9-40364	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)

**Running Applications**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

**Completed Applications**

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

打开spark shell

```
spark-shell
```

```
root@hadoop1:/usr/local/spark-2.2.0/conf# spark-shell
Using Spark's default log4j profile: org/apache/spark/log4j-defaults.properties
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
19/11/19 15:03:21 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classes where applicable
19/11/19 15:03:31 WARN ObjectStore: Version information not found in metastore. hive.metastore.schema.verification is not enabled so recording the schema version 1.2.0
19/11/19 15:03:31 WARN ObjectStore: Failed to get database default, returning NoSuchObjectException
19/11/19 15:03:32 WARN ObjectStore: Failed to get database global_temp, returning NoSuchObjectException
Spark context Web UI available at http://10.173.32.8:4040
Spark context available as 'sc' (master = local[*], app id = local-1574147002287).
Spark session available as 'spark'.
Welcome to

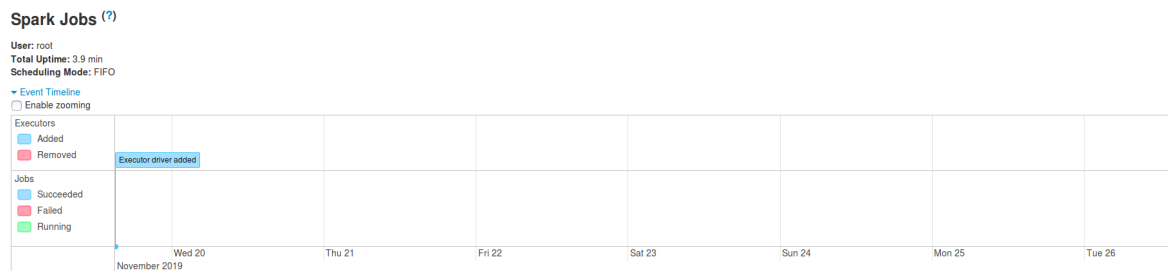
  ____              __
 / ___/____  ____  /  /
/ /  / __ \/ __ \/  /
/___/_/ ___/_/ ___/_/

 version 2.2.0

Using Scala version 2.11.8 (OpenJDK 64-Bit Server VM, Java 1.8.0_222)
Type in expressions to have them evaluated.
Type :help for more information.

scala>
```

访问 `hadoop1:4040`，可以查看当前任务进度



至此，spark已成功安装，并且测试了一些基本功能。

### 3.sbt安装

sbt是用Scala编写的构建工具，可以用来方便地管理依赖。

从官网下载sbt的安装包，随后解压文件，重命名为sbt

```
tar zxvf sbt-1.2.8.tgz
mv sbt-launcher-packaging-0.13.13 sbt
```

在 `/etc/profile` 中配置环境变量，在文件最后添加以下两行

```
export SBT_HOME=/usr/local/sbt
export PATH=$PATH:$SBT_HOME/bin
```

source使profile生效之后，用一下命令查看sbt版本

```
sbt sbtVersion
```

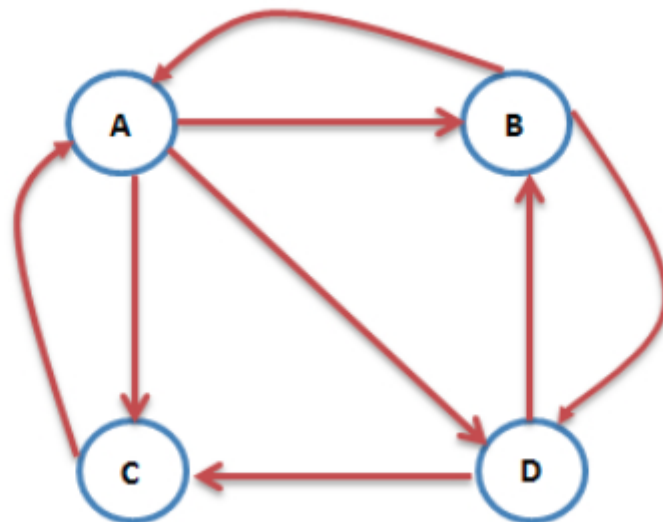
输出版本为1.2.8，表明安装成功！

## 二、PageRank算法

### 1. 算法原理

**简介：**PageRank算法是谷歌提出的，用于计算网页的重要性，当使用搜索引擎搜索网页的时候，越重要的网页越靠前显示。

**基本思想：**将网页抽象为图结点，网页之间的链接关系抽象为有向边，通过定义在这个有向图上的矩阵来计算各个网页之间的**相对重要性**。



以四个节点为例，如果网页A有链接到网页B，则存在一条有向边A→B，上面是一个简单的示例：如果当前在A网页，那么上网者将会各以1/3的概率跳转到B、C、D，这里的3表示A有3条出链，如果一个网页有k条出链，那么跳转任意一个出链上的概率是1/k，同理D到B、C的概率各为1/2，而B到C的概率为0。一般用转移矩阵表示上网者的跳转概率，如果用n表示网页的数目，则转移矩阵M是一个 $n \times n$ 的方阵；如果网页j有k个出链，那么对每一个出链指向的网页i，有 $M[i][j] = 1/k$ ，而其他网页的； $M[i][j] = 0$ 上面示例图对应的转移矩阵如下：

$$M = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix}$$

初试时，假设在每一个网页的概率都是相等的，即1/n，于是初试的概率分布就是一个所有值都为1/n的n维列向量 $V_0$ ，用 $V_0$ 去右乘转移矩阵M，下面是 $V_1$ 的计算过程：

$$V_1 = MV_0 = \begin{bmatrix} 0 & 1/2 & 1 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1/4 \\ 1/4 \\ 1/4 \\ 1/4 \end{bmatrix} = \begin{bmatrix} 9/24 \\ 5/24 \\ 5/24 \\ 5/24 \end{bmatrix}$$

通过 $V_n = MV_{n-1}$ ，不断迭代最终收敛到一组平稳向量。

这其中有些终止点问题和陷阱问题：上述行为是一个马尔科夫过程的实例，要满足收敛性，需要具备一个条件：**图是强连通的，即从任意网页可以到达其他任意网页**

陷阱问题即该节点只存在自己跳到自己的链接，这样不断迭代只会把概率都集中在该节点上，使迭代失去意义。

## 2. 算法实现

为了解决终止点问题和陷阱问题，完整的PageRank算法对此做了改进，将迭代公式转化为：按照经验值 $\alpha = 0.85$ ， $e = 1/N$ ，同时一般来讲，迭代10次左右就结束了。

$$v' = \alpha Mv + (1 - \alpha)e$$

下面是Spark实现的代码：

```
object PageRank {
  def main(args: Array[String]) {
    val inputPath = args(0)
    val iterations = args(1).toInt
    val saveTopNumber = args(2).toInt
    val outputPath = args(3)

    val spark = SparkSession
      .builder
      .appName("PageRank")
      .getOrCreate()

    val lines = spark.read.textFile(inputPath).rdd

    val links = lines
      .map(line => line.split("\\s") match {
        case Array(fromNodeId, toNodeId) => (fromNodeId, toNodeId)
      })
      .distinct()
      .groupByKey()
      .cache()

    var ranks = links.mapValues(_ => 1.0)

    for (_ <- 1 to iterations) {
      val contributions = links
        .join(ranks)
        .flatMap { case (_, (neighbors, rank)) =>
          neighbors.map(neighbor => (neighbor, rank / neighbors.size))
        }

      ranks = contributions
        .reduceByKey(_ + _)
        .mapValues(0.15 + 0.85 * _)
    }

    val output = ranks.takeOrdered(saveTopNumber)
    (Ordering[Double].reverse.on(_._2))
    spark.sparkContext.parallelize(output).saveAsTextFile(outputPath)
    output.foreach(t => println(s"Page: ${t._1}, Rank: ${t._2}"))

    spark.stop()
  }
}
```

```
}
```

### 3. 实际运行

实际运行时，使用sbt对scala代码和spark库进行编译，配置build.sbt的基本引用信息：

- scala的version信息
- spark的基本信息

```
name := "spark-pagerank"
version := "0.1"
scalaVersion := "2.11.7"
libraryDependencies += "org.apache.spark" %% "spark-core" % "2.2.0" % "provided"
libraryDependencies += "org.apache.spark" %% "spark-sql" % "2.2.0" % "provided"
```

ubuntu下运行的终端代码如下：

```
cd ~/spark-pagerank
sbt
compile
sbt package
```

上述命令可生成相应可运行的jar包，然后使用spark-submit运行即可。

目录切换到spark安装目录的bin文件夹下：

```
./spark-submit
--master spark://hadoop1:7077 # 设置master节点
--class PageRank # 设置class包
/root/spark-pagerank/target/scala-2.11/spark-pagerank_2.11-0.1.jar # 包的路径
~/spark-pagerank/web.txt # 输入文件路径
50 # 迭代次数
10 # 前top名次的网页节点
~/spark-pagerank/output.txt #输出文件路径
```

## 三、实验测试

我们进行了两组不同的测试，分别测试不同的迭代次数，和不同的节点个数

### 1.迭代次数

在5个worker节点时，我们分别测试了5次、10次、20次和50次迭代

在web UI可看到每次任务的执行总时间和任务执行情况

URL: spark://hadoop1:7077  
 REST URL: spark://hadoop1:6066 (cluster mode)  
 Alive Workers: 5  
 Cores in use: 10 Total, 0 Used  
 Memory in use: 20.0 GB Total, 0.0 B Used  
 Applications: 0 Running, 4 Completed  
 Drivers: 0 Running, 0 Completed  
 Status: ALIVE

#### Workers

Worker Id	Address	State	Cores	Memory
worker-20191121213846-10.173.32.20-43797	10.173.32.20:43797	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191121213846-10.173.32.22-44687	10.173.32.22:44687	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191121213847-10.173.32.18-46364	10.173.32.18:46364	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191121213847-10.173.32.19-33908	10.173.32.19:33908	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)
worker-20191121213847-10.173.32.21-38267	10.173.32.21:38267	ALIVE	2 (0 Used)	4.0 GB (0.0 B Used)

#### Running Applications

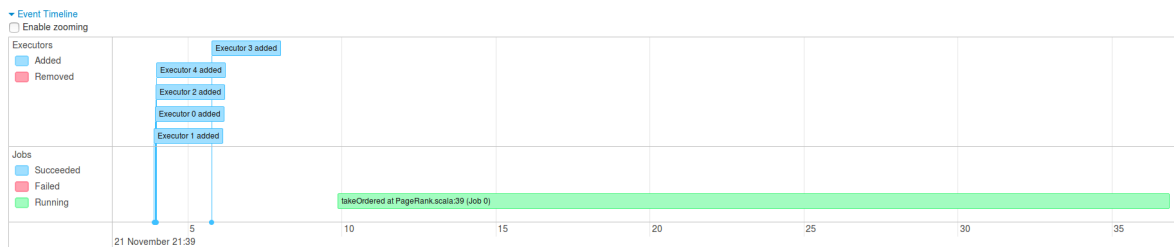
Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
----------------	------	-------	---------------------	----------------	------	-------	----------

#### Completed Applications

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20191121214941-0003	PageRank	10	1024.0 MB	2019/11/21 21:49:41	root	FINISHED	35 s
app-20191121214735-0002	PageRank	10	1024.0 MB	2019/11/21 21:47:35	root	FINISHED	40 s
app-20191121214357-0001	PageRank	10	1024.0 MB	2019/11/21 21:43:57	root	FINISHED	1.6 min

#### Spark Jobs (?)

User: root  
 Total Uptime: 38 s  
 Scheduling Mode: FIFO  
 Active Jobs: 1



#### Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	takeOrdered at PageRank.scala:39	(kill) 2019/11/21 21:39:09	27 s	7/53	62/424

四次运行时间如下：

迭代次数	运行时间
5	35s
10	40s
20	1.3min
50	1.6min

显然，迭代次数越多，程序的运行时间越长，并且程序的IO操作会占用约30s的运行时间

在附录中记录了我们每次程序的输出，我们取前10名的网页。大约20次迭代后，page的排名趋于稳定，系数略有变化。

## 2. 结点数量

将程序的迭代次数固定为50，用不同数量的worker结点来执行MapReduce，运行时间记录如下：

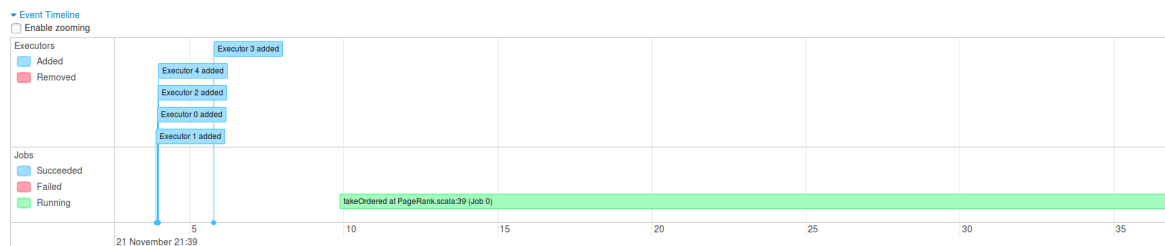
结点数量	运行时间
5	1.6min
4	1.8min
3	1.4min
2	1.8min
1	3.3min



不同结点数量的运行时间并没有明显的规律性（当然相比hadoop时间已大大减小），原因可能是尽管结点个数增加，总的task数量也成比例增加，如下图所示，4个结点时总任务数是424，2个结点变为212，发现**其实每增加一个节点都会相应增加106个task**，因此时间并不一定会减少。当只剩一个节点时，可能由于自身的任务管理更为复杂，总时间有所增加。

#### Spark Jobs (?)

User: root  
Total Uptime: 38 s  
Scheduling Mode: FIFO  
Active Jobs: 1

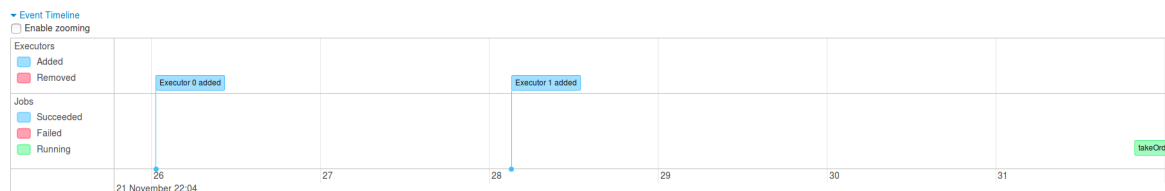


#### Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	takeOrdered at PageRank.scala:39	(kill) 2019/11/21 21:39:09	27 s	7/53	62/424

#### Spark Jobs (?)

User: root  
Total Uptime: 11 s  
Scheduling Mode: FIFO  
Active Jobs: 1



#### Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
0	takeOrdered at PageRank.scala:39	(kill) 2019/11/21 22:04:31	0.1 s	0/53	0/212

在附录中也列出了不同结点的结果，证明只要是迭代次数一致，最终的计算结果是一致的。

## 四、遇到的问题

### 1. Spark的集群运行

需要在每个节点都设置好相应的spark路径信息，写进profile中。

### 2. 运行代码时显示文件找不到

需要在每个节点都复制一份相同路径的文件（web.txt），后将代码文件夹打包传入每一个节点中。

### 3. 无法运行scala含Spark的包

一开始尝试使用下述语句将所有包都引入，但后来程序运行会出错，总是显示job lost，然后报无法assign instance...后续发现是Java反序列化出现相同类方法的错误，解释起来比较复杂没有仔细研究。

```
scala -classpath $(echo *.jar ~/bigdata/spark-2.3.1-bin-hadoop2.7/jars/*.jar| tr ' ' ':') TestRdd.scala
```

后来使用sbt进行配置信息的管理。

### 4. 内存不够大的问题

使用spark的conf对其内存大小进行配置，问题本质上是JVM申请虚拟内存不够的原因。

### 5. 某个节点突然挂掉

重启服务器后得以解决，推测在测试不同结点时，在Spark开启时修改Spark的配置文件，操作不规范导致结点出错。

## 五、组员分工

胡晨旭：算法理解和测试，环境配置

王宇琪：PageRank实现，算法测试

张智为：Spark环境搭建，算法测试

## 六、结果附录

### 1. 20iter 5nodes

Page: 89073, Rank: 3168.183191149973  
Page: 241454, Rank: 2336.7713920943866  
Page: 226411, Rank: 1928.908391829527  
Page: 262860, Rank: 847.0762242546571  
Page: 134832, Rank: 843.566377686055  
Page: 234704, Rank: 695.4173042088398  
Page: 136821, Rank: 687.5316170443247  
Page: 68889, Rank: 681.1086252283856  
Page: 69358, Rank: 664.8763448831974  
Page: 105607, Rank: 652.6235159360374

### 2. 50iter 5nodes

Page: 89073, Rank: 3162.6846596342502  
Page: 241454, Rank: 2325.66809825237  
Page: 226411, Rank: 1912.4882902114828  
Page: 262860, Rank: 844.7906419280941  
Page: 134832, Rank: 843.0198975751777  
Page: 234704, Rank: 691.9762912583044  
Page: 136821, Rank: 687.4011154949313  
Page: 68889, Rank: 680.9790931834499  
Page: 69358, Rank: 664.3026711402322  
Page: 105607, Rank: 649.4780671177593

### 3. 10iter 5nodes

Page: 89073, Rank: 3190.794178228546  
Page: 241454, Rank: 2420.9451316046047  
Page: 226411, Rank: 2008.569604418922  
Page: 134832, Rank: 859.6973674407386  
Page: 262860, Rank: 847.6468032142255  
Page: 234704, Rank: 714.5889310725194  
Page: 136821, Rank: 688.0641293261656  
Page: 68889, Rank: 681.6369860214523  
Page: 69358, Rank: 674.4675486193072  
Page: 105607, Rank: 674.2434713639846

### 4. 5iter 5nodes

Page: 89073, Rank: 3230.8896666527526  
Page: 241454, Rank: 2667.19543227709  
Page: 226411, Rank: 2149.9536649293627  
Page: 134832, Rank: 1065.345018994436  
Page: 262860, Rank: 784.0506475153719

Page: 234704, Rank: 748.9698212000701  
Page: 69358, Rank: 747.7249463743206  
Page: 105607, Rank: 724.9930813348408  
Page: 67756, Rank: 717.0649394321371

#### **5.50iter 4nodes**

Page: 89073, Rank: 3162.6846596342502  
Page: 241454, Rank: 2325.6680982523694  
Page: 226411, Rank: 1912.4882902114828  
Page: 262860, Rank: 844.790641928094  
Page: 134832, Rank: 843.0198975751777  
Page: 234704, Rank: 691.9762912583046  
Page: 136821, Rank: 687.4011154949312  
Page: 68889, Rank: 680.97909318345  
Page: 69358, Rank: 664.3026711402321  
Page: 105607, Rank: 649.4780671177593

#### **6.50iter 3nodes**

Page: 89073, Rank: 3162.6846596342502  
Page: 241454, Rank: 2325.6680982523744  
Page: 226411, Rank: 1912.4882902114798  
Page: 262860, Rank: 844.790641928094  
Page: 134832, Rank: 843.019897575177  
Page: 234704, Rank: 691.9762912583046  
Page: 136821, Rank: 687.4011154949321  
Page: 68889, Rank: 680.9790931834508  
Page: 69358, Rank: 664.3026711402316  
Page: 105607, Rank: 649.478067117759

#### **7.50iter 2nodes**

Page: 89073, Rank: 3162.684659634247  
Page: 241454, Rank: 2325.668098252376  
Page: 226411, Rank: 1912.4882902114805  
Page: 262860, Rank: 844.7906419280937  
Page: 134832, Rank: 843.0198975751771  
Page: 234704, Rank: 691.9762912583049  
Page: 136821, Rank: 687.4011154949329  
Page: 68889, Rank: 680.9790931834513  
Page: 69358, Rank: 664.3026711402324  
Page: 105607, Rank: 649.4780671177592

#### **8.50iter 1node**

Page: 89073, Rank: 3162.6846596342493  
Page: 241454, Rank: 2325.668098252375  
Page: 226411, Rank: 1912.488290211491  
Page: 262860, Rank: 844.790641928096  
Page: 134832, Rank: 843.0198975751782  
Page: 234704, Rank: 691.9762912582989  
Page: 136821, Rank: 687.4011154949314  
Page: 68889, Rank: 680.9790931834492

Page: 69358, Rank: 664.3026711402304  
Page: 105607, Rank: 649.4780671177535