基于Hadoop&Spark的关联规则算法实践总结报告

曾楠嵘

2019/1/16

目 录

1 实践目的及内容

1.1 实践目的

此次实践主要目的在于，希望通过亲身实践，加深自己对 Hadoop、Spark 两类大数据工具的理解，熟悉其从集群部署到运作的基本流程，了解 FPGrowth 算法的基本原理，掌握算法在关联规则中的基本应用，为日后的大数据学习积累经验。

1.2 实践内容

- 搭建 Linux 系统，部署集群；
- 编写 scala 代码，使用 spark 对 Groceries 购物数据进行关联规则的分析；
- 模拟故障，分析 spark 运行情况；

2 实践环境及工具

2.1 实践环境

本次实践主要在个人笔记本电脑与 INTEL NUC 中进行。为方便操作，我在个人笔记本的 Windows 系统中开放 Wifi 热点，NUC 中的 Linux 系统通过 wifi 连接，可与我的 windows 电脑处于同一模拟的局域网内，这样我就可以在个人笔记本中远程登陆 NUC 中的 Linux 集群。

两台机器的性能配置如图 2.1 所示。

图 2.1 实践机器性能配置图

关于实践中具体用到 Hadoop 与 Spark 相关版本如下：

- Hadoop 3.1.1
- JDK 1.8.0_191
- Scala 集群 2.12.8；个人笔记本 2.11.12

- Spark 2.4.0

2.2 实践工具

- **Xshell&Xftp**：用于个人笔记本与 NUC 主机的控制操作及文件传输；
- **VNC**：用于个人笔记本远程登陆 NUC（在实践环境中，对于远程登陆时的命令输入及图形界面操作的延时，VNC 的操作较比 Xshell 要相对流程些，所以选择 VNC 做为主要的远程操作工具）；
- **Virtual Machine Manager**：管理 Linux 中的各 KVM 虚拟机；
- **FinalShell**：用于 NUC 主机与各个虚拟机的控制操作及文件传输；

3 实践步骤

实践步骤主要可分为集群搭建、Hadooph&Spark 环境搭建、程序编写、调试运行、故障模拟五大部分。下面为各部分的具体说明。

3.1 集群搭建

在实践的集群搭建中，是部署的 7 台服务器，一台用作 nameNode,充当 master 角色，其余 6 台用作 dataNode，充当 worker 角色。由于对 secondary NameNode 的相关知识还未进一步接触，暂时未搭建。

根据 NUC 的性能配置情况，对于 nameNode 与各 worker 的性能分配如下：

- master: memory 2G; storage 30G;
- worker:memory 1.5G; storage 30G;

当集群全部运作时，NUC 主机的资源分配情况如图 3.1 所示。

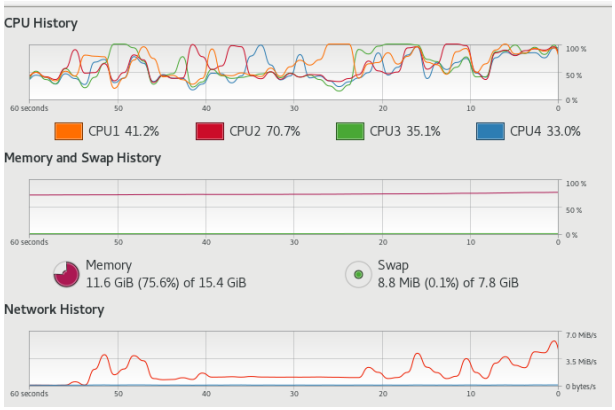


图 3.1 NUC 主机资源分配情况

3.2 Hadoop&Spark 环境搭建

在搭建过程，先是部署的 Hadoop 集群环境，然后在 Hadoop 集群环境的基础上，再部署的 Spark 环境。下面为具体的部署过程及相关配置文档内容。

3.2.1 Hadoop 部署

由于 Hadoop 主要包括 hdfs、yarn、mapReduce 三大模块的部署相对复杂些，下面为大致的部署流程。

- Master 相关
  - 配置 IP，修改/etc/hosts,添加 master 及 6 台 worker DNS 解析；
  - 安装 JDK,配置 java 环境变量
  - 安装 hadoop,配置 hadoop 环境变量
  - 修改$HADOOP_HOME/etc/hadoop/hadoop-env.sh,导入 JAVA_HOME
  - 修改$HADOOP_HOME/etc/hadoop/core-site.xml，设置 fs.defaultFS、hadoop.tmp.dir
  - 修改$HADOOP_HOME/etc/hadoop/hdfs-site.xml,设置 dfs.namenode.secondary.http-address、dfs.replication、dfs.namenode.heartbeat.recheck.interval、dfs.permissions.enabled
  - 修改$HADOOP_HOME/etc/hadoop/mapred-site.xml，设置 mapreduce.framework.name
  - 修改$HADOOP_HOME/etc/hadoop/yarn-site.xml，设置 yarn.resourcemanager.hostname、yarn.nodemanager.aux-services
  - 修改$HADOOP_HOME/etc/hadoop/workers，添加 6 台 worker 的 IP
- 通过 Virtual Machine Manager 复制 Master，生成 6 台 workers
- Workers 相关
  - 配置 IP
- 设置 ssh 免密登陆
- 启动 hadoop 集群，格式化 nameNode

- web 界面访问
  - hadoop 端口 9870
  - yarn 端口 8088

3.2.2 Spark 部署

部署好 hadoop 后，spark 的部署相对简单些。下面为大致的部署流程。

- 安装 scala,配置环境变量
- 安装 spark，配置环境变量
- 修改$SPARK_HOME/conf/**slaves**,添加 6 台 worker 的 IP
- 修改$SPARK_HOME/conf/**spark-env.sh**,设置基本属性
  - JAVA_HOME
  - SCALA_HOME
  - HADOOP_HOME
  - HADOOP_CONF_DIR
  - SPARK_MASTER_IP
  - SPARK_MASTER_PORT
- 修改$SPARK_HOME/conf/**spark-defaults.conf**,设置基本属性
  - spark.yarn.jars
  - spark.master
- 启动 spark 集群$SPARK_HOME/sbin/start-all.sh(必须到 spark 相关目录，否则易于 hadoop 的命令冲突)
- web 页码访问端口 8080

3.3 程序编写

3.3.1 语言选择

虽然 spark 支持 Java、Phthon，但既然 spark 是基于 Scala 编写，同时未来借此实践了解一门新的语言，最后在程序语言上，我选择了 Scala。由于有 Java 和 Python 语言的基础，在 Scala 语言的学习上就显得十分轻松。个人感觉在语法上结合了 Java 和 Python 的优点，语法简洁灵活，是一门不错的语言。

3.3.2 关联规则算法选择

在关联规则算法的选择上，由于 Spark 内部已实现了 FPGrowth 算法，且在效率上，FPGrowth 要由于 Apriori，所以，在程序编写时，我选用 FPGrowth。

3.3.3 关键代码

关键部分代码主要包括数据文件处理、求频繁集、求关联规则三部分，具体的代码如图 3.2 所示。

```
32      val data = sc.textFile(data_path)
33      //对CSV文件进行逗号分割
34      val transactions = data.map(x => x.split(","))
35      transactions.cache()
36      val fpg = new FPGrowth()
37      fpg.setMinSupport(minSupport)
38      fpg.setNumPartitions(numPartitions)
39      val model = fpg.run(transactions)
40      val writer = new PrintWriter(new File("/root/result.txt"))
41
42      writer.println(...)
46      //查看所有的频繁项集，并且列出它出现的次数
47      model.freqItemsets.collect().foreach(itemset => {
48        writer.println(itemset.items.mkString("[", ",", "]") + " : " + itemset.freq)
49      })
50
51      writer.println(...)
55      //通过置信度筛选出推荐规则则
56      model.generateAssociationRules(minConfidence).collect().foreach(rule => {
57        writer.println(rule.antecedent.mkString("[", ",", "]") + "-->" +
58          rule.consequent.mkString("[", ",", "]") + " : " + rule.confidence)
59      })
```

图 3.2　关键代码

3.3.4 调试运行

在 windows 下的 IDEA 中编写好程序代码后，生成任务 jar 包，通过 Xshell 传给 NUC 中的集群试运行。

运行时，主要采取的 Spark 运行模式是 Standalone-client 和 Yarn-client 模式，因为 client 模式可从控制台直接观察任务运行情况。另外，由于该实践仅是针对 Groceries 一个任务，未涉及客户端同时提交多个任务的情况，所以为便于观察任务执行情况，使用 client 模式即可。

4　测试结果及故障模拟

4.1　测试结果

本次实验采用的数据集是 Groceries 数据集。该数据集是某个杂货店一个月真实的交易记录，共有 9835 条消费记录，169 个商品。在测试中，我设定的

minSupport 为 5%，minConfidence 为 30%，在次条件下，通过 FPGrowth 算法，一共得出三条关联规则，具体情况如图 4.1 所示。

```
AssociationRules:
\n----------------------------------------------
[rolls/buns]-->[whole milk] : 0.30790491984521834
[other vegetables]-->[whole milk] : 0.38675775091960063
[yogurt]-->[whole milk] : 0.40160349854227406
```

图 4.1　运行结果

由结果可以看出，该商店在当月内，同时购买 yogurt 和 whole mile 概率最高，相关的还有 rolls/buns 和 other vegetables。所以该超市可以将，yogurt、rolls/buns、other vegetables 的柜架摆在 whole mile 规矩的附件。

4.2　故障模拟

为初步检验 Spark 的稳定性，我简单设置了 3 类故障。令人满意的是，虽然在故障发生时，Spark 运行 groceries 任务的时间有相应增加，但最终的结果和正常情况下的结果一致。

所以，在我实践的任务强度内，Spark 是具有很好的稳定性的。

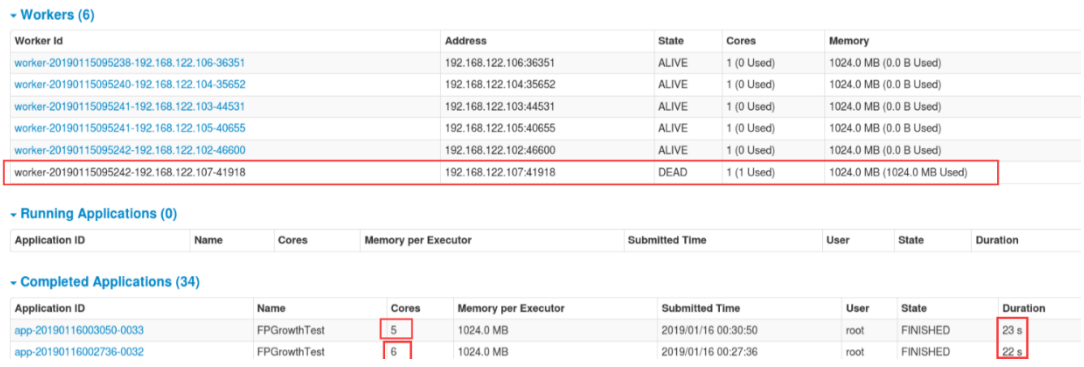
下面为 3 次故障模拟的详细说明。

4.2.1 故障一

故障一是简单模拟集群出现小范围故障的情况。

我在 6 台 worker 全部开启，并执行 groceries 任务的情况下，强制关闭 worker6。通过 Spark 的 web 界面，可以看到，任务的运行几乎没有影响，任务的执行时间和 6 台 workers 正常运行时所用的时间几乎一样。

具体的任务完成情况如图 4.2 所示。



Workers (6)

Worker Id	Address	State	Cores	Memory
worker-20190115095238-192.168.122.106-36351	192.168.122.106:36351	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095240-192.168.122.104-35652	192.168.122.104:35652	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095241-192.168.122.103-44531	192.168.122.103:44531	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095241-192.168.122.105-40655	192.168.122.105:40655	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095242-192.168.122.102-46600	192.168.122.102:46600	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095242-192.168.122.107-41918	192.168.122.107:41918	DEAD	1 (1 Used)	1024.0 MB (1024.0 MB Used)

Running Applications (0)

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

Completed Applications (34)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190116003050-0033	FPGrowthTest	5	1024.0 MB	2019/01/16 00:30:50	root	FINISHED	23 s
app-20190116002736-0032	FPGrowthTest	6	1024.0 MB	2019/01/16 00:27:36	root	FINISHED	22 s

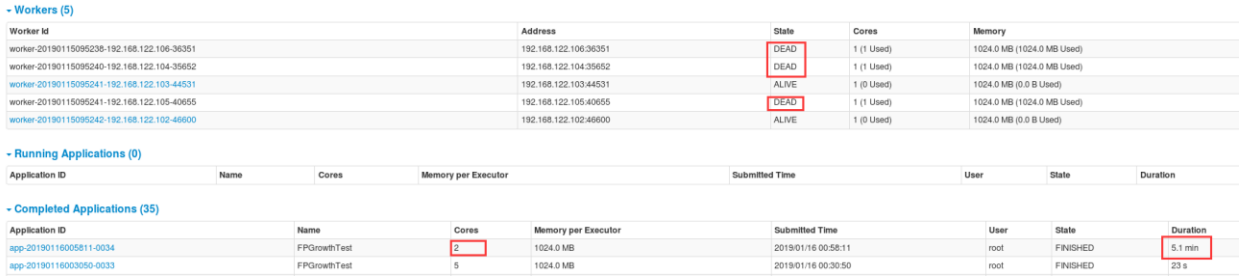
图 4.2 故障一任务完成情况

4.2.2 故障二

故障二是简单模拟集群出现中度范围故障的情况。

我在故障一的基础上，在 5 台 workers 正常运行时，强制关闭其中的 3 台 workers。通过 park 的 web 界面，可以看到，3 台集群的关闭对任务的运行产生了较大影响，任务务的执行时间为 5min,比之前的 23s 高出了 10 倍多的时间，尽管最后的结果是正确的。

具体的任务完成情况如图 4.3 所示。



- Workers (5)

Worker Id	Address	State	Cores	Memory
worker-20190115095238-192.168.122.106-36351	192.168.122.106:36351	DEAD	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20190115095240-192.168.122.104-35652	192.168.122.104:35652	DEAD	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20190115095241-192.168.122.103-44531	192.168.122.103:44531	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)
worker-20190115095241-192.168.122.105-40655	192.168.122.105:40655	DEAD	1 (1 Used)	1024.0 MB (1024.0 MB Used)
worker-20190115095242-192.168.122.102-46600	192.168.122.102:46600	ALIVE	1 (0 Used)	1024.0 MB (0.0 B Used)

- Running Applications (0)

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

- Completed Applications (35)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190116005811-0034	FPGrowthTest	2	1024.0 MB	2019/01/16 00:58:11	root	FINISHED	5.1 min
app-20190116003050-0033	FPGrowthTest	5	1024.0 MB	2019/01/16 00:30:50	root	FINISHED	23 s

图 4.3 故障二任务完成情况

4.2.3 故障三

故障三是简单模拟集群出现大范围故障的情况。

我在 6 台 worker 全部开启，并执行 groceries 任务的情况下，强制关闭 5 台 workers。然而通过 Spark 的 web 界面，看到的却是，5 台集群的关闭对任务的运行产生的影响远小于故障二，任务务的执行时间仅为 1.5min。

于是，仅有一台 worker 的情况，正常运行之前的任务，发现运行的时间仅有 15s,比 6 台 workers 同时运行的时间减少了近一半。

具体的任务完成情况如图 4.4 所示。

- Completed Applications (4)

Application ID	Name	Cores	Memory per Executor	Submitted Time	User	State	Duration
app-20190116014341-0003	FPGrowthTest	1	1024.0 MB	2019/01/16 01:43:41	root	FINISHED	12 s
app-20190116013756-0002	FPGrowthTest	1	1024.0 MB	2019/01/16 01:37:56	root	FINISHED	1.5 min
app-20190116013543-0001	FPGrowthTest	6	1024.0 MB	2019/01/16 01:35:43	root	FINISHED	31 s
app-20190116013434-0000	FPGrowthTest	6	1024.0 MB	2019/01/16 01:34:34	root	FINISHED	36 s

图 4.4 故障三任务完成情况

在网上查阅相关资料后，我初步的猜想是，实践中运行的 groceries 任务量，相比 Spark 的运行能力，所造成的压力是比较轻微的。所示在使用 6 台 workers 运行时，workers 与 master 之间的信息交互占据了较大比重的时间。

另外，我在 spark-default.conf 中对于一些 spark 运行参数，如

spark.default.parallelism，的处理为默认，所以可能没有较大限度的利用当前的集群资源。关于 Spark 参数调优，我将在之后的学习中，深入了解。

5 问题与解决

由于本次实践是我第一次从零开始的、相对完整的进行集群搭建与测试，尽管现在回顾，发现本次实践还是相对简单与轻松的，但在实践过程中，的确遇到了蛮多问题，不过在解决问题的过程中，我学到的知识远远超过了实践本身所需要的知识。

下面我对几个典型的问题情况与解决方案进行简要说明。

5.1 问题一

- 问题描述
  - start-bfs.sh 后，仅 namenode 启动，datanode 未启动
- 问题分析
  - 由于 namenode 和 datanode 中保存的 namespaceID 不同所引起的。可能情况是启动过集群后，又重新执行了 hadoop namenode -format 导致的。
- 解决方案
  - stop-bfs.sh
  - 删掉 datanode 配置的 dfs.data.dir 目录
  - start-bfs.sh
  - 注意：这样一来所有文件就都没有了，要慎重！

5.2 问题二

- 问题描述
  - eyond the 'VIRTUAL' memory limit. 2.2 GB of 2.1 GB virtual memory used.
- 问题分析
  - 为 Yarn 的虚拟内存计算方式导致。任务程序申请的内存为 1G，Yarn 根据此值乘以一个默认比例（2.1），得出申请的虚拟内存的

值。当 Yarn 计算的用户程序所需虚拟内存值大于计算出来的值时，就会报出以上错误。

- 解决方案
  - 在 yarn-site.xml 文件中，将 yarn.nodemanager.vmem-pmem-ratio 的值调高一些，如设为 2.5

5.3 问题三

- 问题描述
  - spark-submit 之后，报 NoClassDefFoundError 异常
- 问题分析
  - 在程序运行时，某些类没有找到
  - 我在$SPARK_HOME/jars 目录下发现，scala-compiler-*.jar 和 scala-library-*.jar 对应用的版本为 2.11.12，而我之前安装的 Scala 版本为最新的 2.12.8，从而找出程序编译与运行的版本不一致。
- 解决方案
  - 在 windows 中下载 Scala 2.11.12,并用其编译原程序

6 总结

本次实践从任务计划的制定到最后的总结报告，大概历时半个月。尽管现在回顾整个实践内容,发现其实并没有我最开始想象的那么困难。但在实践过程中，确实是遇到了许多意想不到的问题，而且有些问题还困扰了蛮久，不过最后都得到了合理的解决，这也是现在最欣慰的了。下面是我在实践过程中一些收获。

- 进一步了解了 Linux 磁盘挂载与分区的相关细节；
- 进一步了解了 Hadoop 中 hdfs 与 yarn 的运行机制；
- 进一步了解了 Spark 的运行模式及相关运行机制；
- 了解了 FPGrowth 关联规则算法；
- 了解了 Scala 语言的基本知识及简单应用；
- 成功从 Eclipse 转到 IDEA；
- 更加熟悉了 GitHub 的使用；

- 熟悉了跨系统的远程管理技巧；

7 参考文章

7.1 集群搭建相关

- 虚拟化技术之 KVM，搭建 KVM
  https://blog.csdn.net/CloudXli/article/details/78306546
- KVM 虚拟机网络配置 Bridge 方式，NAT 方式
  https://blog.csdn.net/hzhsan/article/details/44098537/
- Hadoop3.x 集群搭建
  https://www.cnblogs.com/luhouxiang/p/4829443.html
- Spark2.x 集群部部署
  https://blog.csdn.net/weixin_36394852/article/details/76030317

7.2 关联规则相关

- Spark 调优综述
  https://yq.aliyun.com/articles/461770?spm=a2c4e.11163080.searchblog.114.299d2ec1rNqQ2U
- 频繁项集与关联规则 FP-growth 的原理和实现
  https://blog.csdn.net/huagong_adu/article/details/17739247
  https://www.ibm.com/developerworks/cn/analytics/library/machine-learning-hands-on2-fp-growth/index.html
- 基于 Spark 的 FPGrowth（关联规则算法）
  https://blog.csdn.net/wangqi880/article/details/52910078

7.3 Scala 相关

- Scala 教程
  http://www.runoob.com/scala/scala-tutorial.html
- Scala Standard Library
  https://www.scala-lang.org/api/2.12.8/index.html