第一章　基于 Hadoop 的 MapReduce 实现的性能分析

1.1　Hadoop 调度对性能的影响

Hadoop 程序运行时，会有进程监控节点资源，并将系统中的空闲资源按一定的策略分配给作业，这个进程的核心就是调度器。Hadoop 默认的调度器是 JobQueueTaskScheduler，使用规则为 FIFO (First In First Out，即先入先出) 原则，即按照作业的优先级高低，再按照到达时间的先后选择被执行的作业。

其具体工作流程如下图1.1：

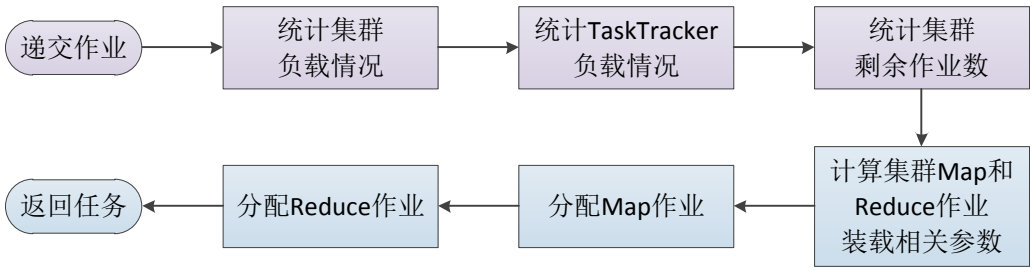

图 1.1 调度器

对于 JobQueueTaskScheduler 的任务调度原则可总结如下：

1. 任务先进先出；
2. 尽量使集群每一个 TaskTracker 达到负载均衡 (这个均衡是 task 数量上的而不是实际的工作强度)；
3. 尽量分配作业的本地任务给 TaskTracker，但不是尽快分配作业的本地任务给 TaskTracker，最多分配一个非本地任务给 TaskTracker(一是保证任务的并发性，二是避免有些 TaskTracker 的本地任务被偷走)，最多分配一个 reduce 任务；
4. 为紧急的 Task 预留一定的 slot；

这种调度方式简单明了，JobTracker 负载较小，适用于大多数的应用场景。

它的主要缺点就是对于在集群比较繁忙的情况，低优先级的作业将很难分配到集群的计算资源，这样对于那些低优先级同时又要求一定的响应时间的短作业来说是非常不利的。

为了解决这种应用差异性带来的性能损失，Hadoop 允许用户自定义调度器。Hadoop 的调度器被设计为一个可插拔的模块，用户可以根据自己实际应用要求设计调度器。

但由于调度机制在单机为分布式下的执行过于简单，其调度结果不具有参考价值，故在本文中不予以进一步的讨论。

1.2 Hadoop 运行机制对 MapReduce 性能的影响

1.2.1 原生 Java

通过第三章的测试结果可以看出，原生 Java 程序运行时，Map 在经过一次大量的读操作之后，其写操作和 Reduce 的读写操作均经过合并优化。Map 的结果被很好的合并，并交给了 Reducer 进行处理。程序运行效率非常高。

由于 Hadoop 提供了类库，所有操作均可通过接口实现，且对 Hadoop 内部透明，方便数据合并及优化，降低了 I/O 压力，性能损失很小。

1.2.2 基于 Streaming 的实现

通过第三章的的测试结果可以看出，基于 Streaming 的实现，Python 的运行时间随着文件的增大逐渐比 Java 长，性能变差的主要是因为 Map 的结果没有进行 combine。从表可以看出，Streaming 中的 Map 和 Reduce 结果没有经过任何 Combine，直接存入本地磁盘，从而造成了大量的空间浪费，增大了 I/O 负担，进入 Reduce 进一步处理的数据没有经过合并，造成性能损失。

以处理 1 亿条日志为例，从表可以得出，Streaming 的实现与 Java 相比，写入量增大约 100 倍，读入量增大约 10 万倍，内存使用量增大约 3 倍，Reduce 函数输入量增大了约 15 万倍，造成了性能损失。

由此可见，MapReduce 框架中的输入输出优化、缓存和合并机制是 MapReduce 程序运行效率的关键。

1.3 节点计算能力对 MapReduce 性能的影响

Hadoop 中，作业在 TaskTracker 上执行，TaskTracker 的计算能力直接影响着 Hadoop 的整体性能。MapReduce 是分布式计算框架，节点之间的通信通过网络传输。在 Map 和 Reduce 过程中，Hadoop 产生了大量的网络和磁盘 I/O，尤其是在 Mapper 和 Reducer 结束时，会引起数据量极大的“网络风暴”，这在实际生产环境中会给网卡和交换机带来很大的压力，若配置不当，将会成为 MapReduce 的瓶颈。

同时，在 Hadoop 运行过程中，Reduce 在 shuffle 阶段对下载来的 Map 数据，并不是立刻就写入磁盘的，而是会先缓存在内存中，然后当使用内存达到一定量的时候才刷入磁盘，这就要求节点机有足够大的内存去缓存数据，默认情况

下，reduce 会使用其 heapsize 的 70% 来在内存中缓存数据。内存的大小决定了缓存数量的多少，减少不必要的 I/O。

磁盘读写速度也是 MapReduce 中影响性能的一个重要指标，在 Map 阶段，Hadoop 会将 Map 的中间结果保存在节点本地磁盘，因为中间结果是临时的，不需要写多份。本地磁盘的读写 I/O 速度在某些应用中也会成为 MapReduce 程序的瓶颈。

另外，在实际生产环境中，节点计算之间的计算能力会有略微的不同，默认的 Hadoop 调度器会使作业运行时出现性能短板，这种短板在异构环境中尤为明显。

1.4 实例程序编写方法对 MapReduce 性能的影响

通过 Java 和 Streaming 两种实现方式的比较，可以看出影响 MapReduce 较大的部分为 Suffle 部分，从 MapReduce 运行原理可以看出，Suffle 部分的合并是根据 Key 值合并的，如何合理的选择 Key 值，降低 Reducer 的处理时间，对 MapReduce 程序的运行起着至关重要的作用。

但实际开发中，Key 的分布并不一定是均匀的，极端情况下，所有数据共享一个 Key，后果是 reduce 任务只会在集群其中一个节点上运行，不仅没有利用好集群处理能力，反倒因为大量数据集中而导致计算效率低下，降低程序整体运行效率。

作为了解决这个问题，程序员需要在编程时尽量选择分布均匀的参数作为 Key，或者使用其他方法将 Key 打散。下文以 Web 日志处理中的处理 UV 为例，描述了一种打散 Key 的方法。

UV 是独立访客 (unique visitor) 的缩写，对于网站的用户粘度和回头率统计有着至关重要的作用。

在网站设计中，通常会给 Cookie 一个 Key 为 uid，Value 为用户 id 的一组数据。在 Web 日志中，该 uid 即可用来统计 UV。

在未打散 Key 的情况下，uid 的值会比较分散，但对于游客用户来说，uid 值为空，即 0，这就会造成 Map 的结果中，uid 分布不均匀。这时，将 uid 的值设为用户 ip 的后三位，从而使游客的统计均匀的分发到不同的 Key 中。但仅仅这样做是不够的，每个 uid 都不同，会导致 Map 结果中 Key 的数量过多，不便于数据合并和传输。这时，可以使用取模函数对 uid 取模，将 Key 的数量控制在合适的范围之内。进而分发到不同的 Reduce 节点上。

在 Reduce 阶段，Reducer 接收到 Key/Value 对时，可以对 Cookie 值进行判断，若 uid 为空，则访客数 +1，反之则该 uid 的统计数 +1。

具体实现流程如下图：

实际生产环境中，Key 的打散和控制还有其他多种方法，如将字符串转换为整形数字提高处理速度、或者将一个 MapReduce 操作划分成两份 MapReduce 程序去执行。具体使用方法应该根据应用程序的要求来确定，在此不一一赘述。

1.5 本章小结

第二章　总结与展望

2.1　本文总结

本文通过对基于 Hadoop 实现的 MapReduce 编程框架的介绍，详述了 MapReduce 编程思想及 Hadoop 的实现过程，并在此基础上使用两种实现方式实现了 Web 访问日志程序的编写与其性能分析。

根据第三章的测试和第四章的分析，可以得出以下四个结论：

1. MapReduce 运行效率高，分布式程序编写简单，是个优秀的编程框架。
2. I/O 可能会造成 MapReduce 性能瓶颈
3. 现阶段，原生 Java 比 Streaming 效率高
4. MapReduce 程序编写中，Key 的选择对数据合并、Reduce 分发有着至关重要的作用。

2.2　进一步的工作

1. 使用不同的 Hadoop 调度机制，测试其对集群的影响，找出并总结其调度方式所对应的应用场景。
2. 找出 Streaming 中负责数据合并的部分，理解其处理过程并进行适当优化。

附录 A　实验环境搭建

Hadoop 使用 Java 语言实现，集群之间以 ssh 协议通信，故 Hadoop 的依赖较少，且安装较为简便，其大体过程为：安装操作系统，安装 ssh 和 jdk，解压缩 Hadoop 二进制包，编辑 Hadoop 配置文件并设置 ssh、jdk 和 hadoop 环境变量。本文以具体过程如下：

1. 最小化安装 centos_6.2_x86-64

2. 设置网络，并将源设置为西电源，关闭防火墙

```
iptables -F
iptables -X
iptables -Z
iptables-save > /etc/sysconfig/iptables
```

3. 更新系统

```
yum upgrade -y
```

4. 安装依赖包 rsync openssh-server openssh-clients

```
yum install -y rsync openssh-server openssh-clients
```

5. 下载 sun-jdk 和 hadoop 包 6. 安装 jdk 到/usr/lib 并创建相应软链接

```
cp jdk-6u31-linux-x64.bin /usr/lib
cd /usr/lib
sh jdk-6u31-linux-x64.bin
rm jdk-6u31-linux-x64.bin
ln -s jdk1.6.0_31 jdk
```

7. 创建 hadoop 用户和用户组，并设置密码

```
groupadd hadoop
useradd -g hadoop hadoop -d /home/hadoop
passwd hadoop
```

8. 创建 ssh 密钥公钥，并加入可信列表

```
su hadoop
ssh-keygen -t rsa -P ""
cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

验证密钥登录是否生效

```
ssh localhost
```

若没生效，检查文件权限

```
chmod 644 ~/.ssh/authorized_keys
```

9. 解压缩 hadoop-1.0.1 到/usr/local 并创建软链接

```
su
cp hadoop-1.0.1.tar.gz /usr/local
cd /usr/local
tar zxvf hadoop-1.0.1.tar.gz
ln -s hadoop-1.0.1 hadoop
chown hadoop:hadoop hadoop-1.0.1 hadoop -R
```

10. 给 hadoop 用户设置环境变量

```
vi ~/.bashrc
```

文件末尾添加环境变量如下：

```
# Set Hadoop-related environment variables
export HADOOP_HOME=/usr/local/hadoop
# Set JAVA_HOME (we will also configure JAVA_HOME directly for Hadoop later on)
export JAVA_HOME=/usr/lib/jdk
# Add Hadoop bin/ directory to PATH
export PATH=$PATH:$HADOOP_HOME/bin
```

重新登录使变量生效 11. 编辑 hadoop 配置 (/usr/local/hadoop/conf)

```
conf/hadoop-env.sh
```

替换

```
# export JAVA_HOME=/usr/lib/j2sdk1.5-sun
```

为

```
export JAVA_HOME=/usr/lib/jdk
```

添加

```
# Disable the HADOOP_HOME_WARN
export HADOOP_HOME_WARN_SUPPRESS=TRUE
```

取消 hadoophome 设置警报 core-site.xml

```
<configuration>
     <property>
         <name>fs.default.name</name>
         <value>hdfs://localhost:9000</value>
     </property>
</configuration>
```

hdfs-site.xml

```
<configuration>
     <property>
         <name>dfs.replication</name>
         <value>1</value>
     </property>
</configuration>
```

mapred-site.xml

```
<configuration>
     <property>
         <name>mapred.job.tracker</name>
         <value>localhost:9001</value>
     </property>
</configuration>
```

12. 格式化 hdfs

```
hadoop namenode -format
```

13. 启动 hadoop

```
start-all.sh
```

运行命令 jps，查看 Hadoop 进程启动情况。如果 TaskTracker、DataNode、Jps、NameNode、SecondaryNameNode 和 JobTracker 子进程均成功启动，则 Hadoop 运行正常。至此，Hadoop 就安装完成了。

致　谢

毕业论文暂告收尾，这也意味着我在西安电子科技大学的四年的学习生活既将结束。回首既往，自己一生最宝贵的时光能于这样的校园之中，能在众多学富五车、才华横溢的老师们的熏陶下度过，实是荣幸之极。在这四年的时间里，我在学习上和思想上都受益非浅。这除了自身努力外，与各位老师、同学和朋友的关心、支持和鼓励是分不开的论文的写作是枯燥艰辛而又富有挑战的。

数学是理论界一直探讨的热门话题，老师的谆谆诱导、同学的出谋划策及家长的支持鼓励，是我坚持完成论文的动力源泉。在此，我特别要感谢我的导师 xxx 老师。从论文的选题、文献的采集、框架的设计、结构的布局到最终的论文定稿，从内容到格式，从标题到标点，她都费尽心血。没有 xxx 老师的辛勤栽培、孜孜教诲，就没有我论文的顺利完成。

感谢数学系的各位同学，与他们的交流使我受益颇多。最后要感谢我的家人以及我的朋友们对我的理解、支持、鼓励和帮助，正是因为有了他们，我所做的一切才更有意义；也正是因为有了他们，我才有了追求进步的勇气和信心。

时间的仓促及自身专业水平的不足，整篇论文肯定存在尚未发现的缺点和错误。恳请阅读此篇论文的老师、同学，多予指正，不胜感激！.....

谨把本文献给我最敬爱的父母亲以及所有关心我的人！

参考文献

[1] http://news.netcraft.com/archives/2012/05/02/may-2012-web-server-survey.html

[2] CTEX 翻译小组. lshort 中文版 . (2003)

[3] 邓建松, 彭冉冉, 陈长松. LATEX 2ε 科技排版指南. 科学出版社, 书号: 7-03-009239-2/TP.1516, 北京, (2001).

[4] 王磊. LATEX 2ε 插图指南. (2000).

[5] 张林波. 于新版 CCT 的说明. (2003).

[6] Donald E, Knuth. Computer Modern Typefaces, volume E of Computers and Typesetting. Addison-Wesley, Reading, Massachusetts, (1986).

[7] Donald E. Knuth. METAFONT: The Program, volume D of Computers and Typesetting. Addison-Wesley, Reading, Massachusetts, (1986).

[8] Donald E. Knuth. The METAFONTbook, volume C of Computers and Typesetting. Addison-Wesley, Reading, Massachusetts, (1986).

[9] Donald E. Knuth. TEX: The Program, volume B of Computers and Typesetting. Addison-Wesley, Reading, Massachusetts, (1986).

[10] Donald E. Knuth. The TeXbook, volume A of Computers and Typesetting. Addison-Wesley, Reading, Massachusetts, (1986).

[11] Leslie Lamport. LATEX — A Document Preparation System: User's Guide and Reference Manual. Addison-Wesley, Reading, Massachusetts, 2nd edition, (1985).