# 实验四：Spark Streaming

## 一、 实验目的

- 学习 Spark Streaming 分布式处理框架，理解流式数据处理概念
- 实现简单流式数据产生、接收与处理。

## 二、 任务与要求

- 使用 netcat 指令产生测试流数据
- 使用 Spark Streaming 接收流式数据
- 使用 Spark Streaming 完成词频统计任务
- （加分项）实现流式数据的安全退出机制（例如检测到关键词后退出流式处理程序）

## 三、 实验原理

### 3.1. Spark Streaming 概述

Spark Streaming 是核心 Spark API 的扩展，可实现可扩展、高吞吐量、可容错的实时数据流处理。数据可以从诸如 Kafka，Flume，Kinesis 或 TCP 套接字等众多来源获取，并且可以使用由高级函数（如 map，reduce，join 和 window）开发的复杂算法进行流数据处理。最后，处理后的数据可以被推送到文件系统，数据库和实时仪表板。而且，您还可以在数据流上应用 Spark 提供的机器学习和图处理算法。


在内部，它的工作原理如下。Spark Streaming 接收实时输入数据流，并将数据切分成批，然后由 Spark 引擎对其进行处理，最后生成"批"形式的结果流。


Spark Streaming 将连续的数据流抽象为 discretized stream 或 DStream。 可以从诸如 Kafka，Flume 和 Kinesis 等来源的输入数据流中创建 DStream，或者通过对其他 DStream 应用高级操作来创建。在内部，DStream 由一个 RDD 序列表示。

# 四、 实验步骤

### 4.1 任务 1. 使用 netcat 指令产生测试数据流

netcat 是网络工具中的瑞士军刀，它能通过 TCP 和 UDP 在网络中读写数据。通过与其他工具重定向，可以实现很多复杂的功能，如端口扫描、流式数据生成或保存、文件传输等。本次实验主要用 netcat 生成流式数据。

首先，我们需要开启两个终端，一个用于生成流式数据，另一个用于接收流式数据。在两个窗口中分别执行下面的指令（把 11009 改成学号的后四位或五位，避免端口冲突）：

```
thumm01:~$ nc localhost 11009
```

```
thumm01:~$ nc -lk 11009
```

接着在任意一端输入字符串，这个字符串会被这一端的 netcat 以流数据的形式发送到对应的端口，并在另一端被 netcat 接收并输出。（这是一个最简单的聊天软件☺）

```
thumm01:~$ nc localhost 11009
Hello World
```

```
thumm01:~$ nc -lk 11009
Hello World
```

此外，netcat 还可以与重定向符号">"结合直接将文件数据转成网络流发送到另一端：

```
thumm01:~$ vim test.txt
thumm01:~$ cat test.txt
spark streaming test
thumm01:~$ nc localhost 11009 < test.txt
```

```
thumm01:~$ nc -lk 11009
spark streaming test
```

如果右边改成 nc -lk 11009 > test_recvd.txt 就可以把接受的流保存成文件，完成了一次文件传输。

### 4.2 任务 2. 使用 Spark Streaming 接收流数据

前面使用了 netcat 接收流数据并打印，接下来使用 SparkStream 接收流数据，为后续流数据处理做准备。

首先使用 netcat 交互地生成流式数据:

```
thumm01:~$ nc -lk 11009
```

然后在另一个终端启动 spark-shell:

```
thumm01:~$ spark-shell
//spark-shell 启动信息
```

```
scala> import org.apache.spark.streaming.{Seconds, StreamingContext}  //导入相关 SparkStreaming 包
import org.apache.spark.streaming.{Seconds, StreamingContext}

scala> val ssc = new StreamingContext(sc, Seconds(1))   // 创建流式
ssc: org.apache.spark.streaming.StreamingContext = org.apache.spark.streaming.StreamingContext@2114955c

scala> val lines = ssc.socketTextStream("thumm01", 11009)   //监听 thumm01 的 11009 端口
lines: org.apache.spark.streaming.dstream.ReceiverInputDStream[String] =
org.apache.spark.streaming.dstream.SocketInputDStream@105fa381

scala> lines.print()    //打印接收到的信息
scala> ssc.start()      //启动流数据接受与处理
... //每隔一段时间刷新一次,如果接收到信息就输出，没有接受就只输出时间
scala> ssc.stop ()      //停止流数据接收
```

在 netcat 端输入"hello world", 回车，在 spark-shell 端会出现下面的输出，表示接收到信息:

```
-------------------------------------------
Time: 1575733886000 ms
-------------------------------------------
hello world          <= 接收到的信息
 [Stage 2:> (0 + 1) / 1]-----------------------------------------
```

Spark Streaming 需要输入 ssc.stop( )停止流数据的接收，而 ctrl + c 不会让程序退出，这点需要注意~

从上面的流程，我们可以看到 Spark Streaming 处理流式数据的过程是先定义处理流程，然后启动任务，这点和 Tensorflow 的计算图类似。

### 4.3 任务 3. 使用 Spark Streaming 做词频统计

当我们能接收到流数据以后，下一步就是对流数据进行处理，这里还是做最经典的词频统计任务。

在一个终端启动 spark-shell，输入下面指令：

```
import org.apache.spark.streaming._                  //导入数据包
val ssc = new StreamingContext(sc, Seconds(5))       //创建流数据上下文，每隔 5 秒创建一个流式 RDD
val lines = ssc.socketTextStream("thumm01", 11009)  //监听 thumm01 的端口
val result = lines.flatMap(_.split(" ")).map(w => (w, 1)).reduceByKey(_ + _)   //词频统计
result.print()                                        //输出词频统计结果
ssc.start()                                           //启动任务
// ssc.stop() 最后退出的时候输入
```

**在指令执行过程中，程序会输出很多错误，这个可以不用管，因为生成流数据的那端还没开。**

在另一个终端产生数据，这里使用 netcat 将 wc_dataset.txt 作为数据源生成流：

```
thumm01:~$ nc -l 11009 < /home/dsjxtjc/wc_dataset.txt
```

这个时候，Spark Streaming 端就会显示词频统计的结果：

```
-------------------------------------------
Time: 1575736385000 ms
-------------------------------------------
(multiplication,33)
```

```
(young,163)

(someone,33)

...

[Stage 2:>  (0 + 1) / 1]
```

这个时候我们就完成了统计 5 秒内词频的功能，对这个程序做一定的修改，就能实现一个统计累计词频的功能。如果需要保存结果的话，可以用 SaveAsTextFile，也可以设置 Checkpoint:

SaveAsTextFile:

```
scala> result.saveAsTextFiles("/dsjxtjc/2018211009/wc_sc_out")          //每次计算保存结果
```

Checkpoint:

```
scala> ssc.checkpoint("/dsjxtjc/2018211009/wc_sc_ckpt")                 //将每个时间片的结果保存
```

需要查看结果只需要将结果传到本地。

# 五、 作业提交要求

- 在词频统计任务中，要求大家：
  - 生成连续的流数据，统计从任务开始到某一时间点的**累计**词频（任务 3 只实现某一时间片的词频统计），使用 checkpoint 保存不同时间段的结果，并贴出不同时间点累计词频前 10 的词。
  - （加分项）实现流式数据的安全退出机制，例如检测到流中有关键词 **:exit** 退出流处理程序。
- 将以上作业所用到的数据、命令、步骤截图等写入实验报告，然后连同所有代码文件一同打包成压缩文件，上交至网络学堂。
- 迟交作业一周以内，以 50%比例计分；一周以上不再计分。另一经发现抄袭情况，零分处理。
- 助教联系方式：严浩鹏（yhp18@mails.tsinghua.edu.cn）
- 成绩计算：**基础分满分 10 分，加分项满分 2 分。**

# 附录：

有些同学期末大作业做流式数据处理的可以使用下面的程序产生流式数据。该程序将文件转成可以被 SparkStreaming 接收的流数据，并每隔一段时间发送一部分内容：

```
import sys
import socket
import time

host = "0.0.0.0"          # 监听本机地址，
port = int(sys.argv[1])   # 参数 1 为监听端口，如 11009
file_path = sys.argv[2]   # 参数 2 为数据源地址，如/home/dsjxtjc/wc_dataset.txt

sock_fd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)   # 创建流式 socket
sock_fd.bind((host, port))                                    # 绑定端口，准备监听
sock_fd.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1) # 设置参数用于正常关闭

sock_fd.listen(5)                                             # 设置最大监听数
print("Start listening~")

conn, addr = sock_fd.accept()                                 # 一旦有连接则返回连接描述符和地址
print("Connected by", addr)

try:
    fp = open(file_path)                                      # 打开数据源 文件
    lines = fp.readlines()                                    # 读取数据源 文件
    for line in lines:
      # 每隔一段时间将部分数据发送给接收到，这里按行发送，也可以每次发送多行
      conn.send(line.encode('utf-8'))
       time.sleep(0.2)
except socket.error:
    print ('Error Occured.\n\nClient disconnected.\n')
finally:
    fp.close()

sock_fd.shutdown(socket.SHUT_RDWR)                            # 关闭连接
sock_fd.close()
```