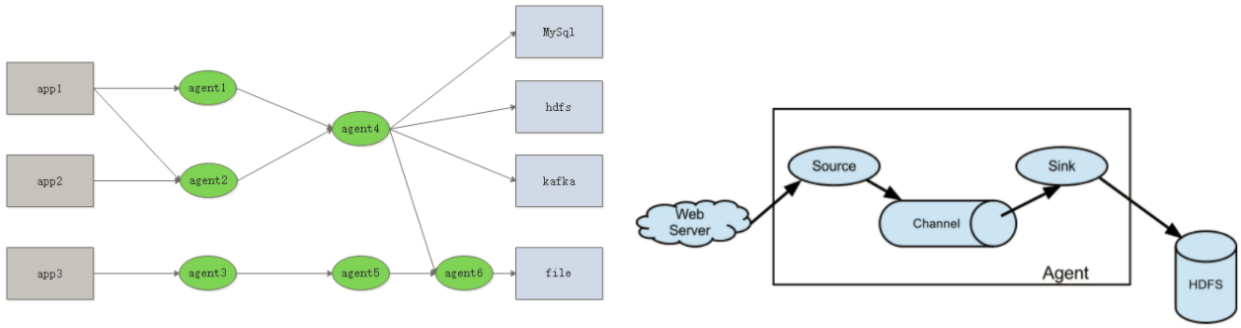
实时点击流日志处理



flume的核心是agent，而agent包含source、channel、sink三个组件。

- source：source组件是专门用来收集数据的，可以处理各种类型、各种格式的日志数据,包括avro、thrift、exec、jms、spooling directory、netcat、sequence generator、syslog、http、legacy、自定义。
- channel：source组件把数据收集来以后，临时存放在channel中，即channel组件在agent中是专门用来存放临时数据的——对采集到的数据进行简单的缓存，可以存放在memory、jdbc、file等等。
- sink：sink组件是用于把数据发送到目的地的组件，目的地包括hdfs、logger、avro、thrift、ipc、file、null、hbase、solr、自定义

工作流程：flume把数据从数据源(source)收集过来，再将数据送到指定的目的地(sink)。为保证传输的过程一定成功，在送到目的地(sink)之前，会先缓存数据(channel),待数据真正到达目的地(sink)后，flume再删除自己缓存的数据。 在整个数据的传输的过程中，流动的是event（基本单位），因此事务保证是在event级别进行的。

event：event将传输的数据进行封装，是flume传输数据的基本单位，如果是文本文件，通常是一行记录。event也是事务的基本单位。event在单个agent中经历source—channel—sink过程，后面可能输出到下一个agent或者flume外的系统中。event本身为一个字节数组，其携带headers(头信息)信息，消息体，消息内容

从上面图中可以看出flume支持多级的成网状数据流动，非常的灵活好用，这应该就是flume广泛使用原因吧。比如数据扇入到同一个agent或者扇出到多个agent。

flume配置

编辑： `/root/bigdata/flume/conf/click_trace_log_hdfs.properties`

```
# Name the components on this agent
a1.sources = r1
a1.sinks = k1 k2
a1.channels = c1

# Describe/configure the source
a1.sources.r1.type = exec
a1.sources.r1.command = tail -F /root/meiduoSourceCode/logs/click_trace.log
a1.sources.r1.channels = c1

a1.sources.r1.interceptors = t1
a1.sources.r1.interceptors.t1.type = timestamp

# Use a channel which buffers events in memory
a1.channels.c1.type = memory
a1.channels.c1.capacity = 1000
a1.channels.c1.transactionCapacity = 100

a1.sinks.k1.type = hdfs
a1.sinks.k1.channel = c1
a1.sinks.k1.hdfs.path = hdfs://localhost:9000/project2-meiduo-rs/logs/click-trac
e/%y-%m-%d
as.sinks.k1.hdfs.userLocalTimeStamp = true
a1.sinks.k1.hdfs.filePrefix = click-trace-
a1.sinks.k1.hdfs.fileType = DataStream
a1.sinks.k1.hdfs.writeFormat = Text
a1.sinks.k1.hdfs.round = true
a1.sinks.k1.hdfs.roundValue = 10
a1.sinks.k1.hdfs.roundUnit = minute

a1.sinks.k2.channel = c1
a1.sinks.k2.type = org.apache.flume.sink.kafka.KafkaSink
a1.sinks.k2.kafka.topic = meiduo_click_trace
a1.sinks.k2.kafka.bootstrap.servers = localhost:9092
a1.sinks.k2.kafka.flumeBatchSize = 20
a1.sinks.k2.kafka.producer.acks = 1
a1.sinks.k2.kafka.producer.linger.ms = 1
a1.sinks.k2.kafka.producer.compression.type = snappy
```

启动flume对点击流日志进行采集，分别发送到kafka和hdfs： `flume-ng agent -f /root/bigdata/flume/conf/click_trace_log_hdfs.properties -n a1`

启动Kafka(如果还未启动的话)： `cd /root/bigdata/kafka && bin/zookeeper-server-start.sh -daemon config/zookeeper.properties && bin/kafka-server-start.sh config/server.properties`

In [1]:
```
import os
# 配置pyspark和spark driver运行时 使用的python解释器
JAVA_HOME = '/root/bigdata/jdk'
PYSPARK_PYTHON = '/miniconda2/envs/py365/bin/python'
# 当存在多个版本时，不指定很可能会导致出错
os.environ['PYSPARK_PYTHON'] = PYSPARK_PYTHON
os.environ['PYSPARK_DRIVER_PYTHON'] = PYSPARK_PYTHON
os.environ['JAVA_HOME'] = JAVA_HOME
# 注意，如果是使用jupyter或ipython中，利用spark streaming链接kafka的话，必须加上下面语
# 同时注意：spark version>2.2.2的话，pyspark中的kafka对应模块已被遗弃，因此这里暂时只能
os.environ["PYSPARK_SUBMIT_ARGS"] = "--packages org.apache.spark:spark-streaming-kafka
# 配置spark信息
from pyspark import SparkConf
import pyspark

SPARK_APP_NAME = "meiduo_logs"
SPARK_URL = "spark://192.168.58.100:7077"

conf = SparkConf()    # 创建spark config对象
config = (
    ("spark.app.name", SPARK_APP_NAME),    # 设置启动的spark的app名称，没有提供，将随机
    ("spark.executor.memory", "2g"),    # 设置该app启动时占用的内存用量，默认1g，指一
    ("spark.master", SPARK_URL),    # spark master的地址
    ("spark.executor.cores", "2"),    # 设置spark executor使用的CPU核心数，指一台虚拟机
#     ("hive.metastore.uris", "thrift://localhost:9083"),    # 配置hive元数据的访问，
#
    # 以下三项配置，可以控制执行器数量
#     ("spark.dynamicAllocation.enabled", True),
#     ("spark.dynamicAllocation.initialExecutors", 1),    # 1个执行器
#     ("spark.shuffle.service.enabled", True)
#   ('spark.sql.pivotMaxValues', '99999'),  # 当需要pivot DF，且值很多时，需要修改，默
)
# 查看更详细配置及说明：https://spark.apache.org/docs/latest/configuration.html

conf.setAll(config)

# 利用config对象，创建spark session
sc = pyspark.SparkContext(master=SPARK_URL, conf=conf)
```

In [2]:
```
# 注意：初次安装并运行时，由于使用了kafka，所以会自动下载一系列的依赖jar包，会耗费一定

from pyspark.streaming.kafka import KafkaUtils
from pyspark.streaming import StreamingContext

# 第2个参数表示 the time interval (in seconds) at which streaming data will be divided
ssc = StreamingContext(sc, 0.5)

kafkaParams = {"metadata.broker.list": "192.168.58.100:9092"}
dstream = KafkaUtils.createDirectStream(ssc, ["meiduo_click_trace"], kafkaParams)
```

...

In [29]:
```
import re
def map(row):
    match = re.search("\
exposure_timesteamp<(?P<exposure_timesteamp>.*?)> \
exposure_loc<(?P<exposure_loc>.*?)> \
timesteamp<(?P<timesteamp>.*?)> \
behavior<(?P<behavior>.*?)> \
uid<(?P<uid>.*?)> \
sku_id<(?P<sku_id>.*?)> \
cate_id<(?P<cate_id>.*?)> \
stay_time<(?P<stay_time>.*?)>", row[1])

    result = []
    if match:
        result.append(("exposure_timesteamp", match.group("exposure_timesteamp")))
        result.append(("exposure_loc", match.group("exposure_loc")))
        result.append(("timesteamp", match.group("timesteamp")))
        result.append(("behavior", match.group("behavior")))
        result.append(("uid", match.group("uid")))
        result.append(("sku_id", match.group("sku_id")))
        result.append(("cate_id", match.group("cate_id")))
        result.append(("stay_time", match.group("stay_time")))
    return result

def foreachRDD(rdd):
    print("foreachRDD", rdd.collect())
```

In [30]:
```
dstream.map(map).foreachRDD(foreachRDD)
```

In [31]:
```
ssc.start()
```

...

In [32]:

```
# 生成日志
import logging#log: 记录
import time

def get_logger(logger_name, path, level):

    # 创建logger
    logger = logging.getLogger(logger_name)
    # level:  OFF、FATAL、ERROR、WARN、INFO、DEBUG、ALL或者自己定义的级别
    logger.setLevel(level)

    # 创建formatter
    # %(asctime)s: 打印日志的时间
    # %(message)s: 打印日志信息
    fmt = '%(asctime)s: %(message)s'
    datefmt = '%Y/%m/%d %H:%M:%S'
    formatter = logging.Formatter(fmt,datefmt)

    # 创建handler
    # FileHandler: writes formatted logging records to disk files
    handler = logging.FileHandler(path)
    handler.setLevel(level)

    # 添加handler和formatter 到 logger
    handler.setFormatter(formatter)
    logger.addHandler(handler)

    return logger

click_trace_logger = get_logger('click_trace','/root/workspace/3.rs_project/project2/r
                                logging.DEBUG)

# 点击流日志
exposure_timesteamp = time.time()
exposure_loc = 'detail'
timesteamp = time.time()
behavior = 'pv' # pv fav cart buy
uid = 1
sku_id = 1
cate_id = 1
stay_time = 60
# # 假设某点击流日志记录格式如下:
click_trace_logger.info("exposure_timesteamp<%d> exposure_loc<%s> timesteamp<%d> behav
                        %(exposure_timesteamp, exposure_loc, timesteamp, behavior, ui

```

```
foreachRDD []
foreachRDD [[('exposure_timesteamp', '1608781198'), ('exposure_loc', 'detail'), ('tim
esteamp', '1608782037'), ('behavior', 'pv'), ('uid', '1'), ('sku_id', '1'), ('cate_i
d', '1'), ('stay_time', '60')]]
foreachRDD []
foreachRDD []
foreachRDD []
foreachRDD []
foreachRDD []
```

```
foreachRDD []
foreachRDD []
foreachRDD []
foreachRDD []
foreachRDD []
foreachRDD [[('exposure_timesteamp', '1608781198'), ('exposure_loc', 'detail'), ('tim
esteamp', '1608782037'), ('behavior', 'pv'), ('uid', '1'), ('sku_id', '1'), ('cate_i
d', '1'), ('stay_time', '60')], [('exposure_timesteamp', '1608781198'), ('exposure_lo
c', 'detail'), ('timesteamp', '1608782037'), ('behavior', 'pv'), ('uid', '1'), ('sku_
id', '1'), ('cate_id', '1'), ('stay_time', '60')], [('exposure_timesteamp', '16087811
```

In [ ]:
```
1 ssc.stop()
```