本项目是学院金融知识图谱项目的一个部分，需求是通过分布式爬虫实时爬取主要金融网站与论坛的金融资讯，并且前端提供一个界面，用户输入爬取的url，爬取规则与爬取优先级，使其可以手动添加爬取网站。爬取的数据一方面进行持久化，保存在hdfs中，进行历史信息的保存（进行产业上下游分析与信息抽取，不属于本项目范围），另一方面进行实时信息分析（如pagerank，elasticsearch不属于本项目范围）。基于以上需求，我们组进行了调研，初步确定了以下几个模块：Scrapy-redis flume kafka storm hbase hadoop zookeeper

下面是调研的结果：

爬虫：Scrapy-redis：

根据需求，我们需要爬取主流金融网站与论坛，爬取内容为新闻，为文本，所以基本不需要考虑图片等，对于需要异步加载或者用户登录的网站，我们初步决定简单的进行放弃（学院内部讨论结果，后面可能会对特定的网站使用专门的爬取策略，目前只是一个粗糙的通用的爬虫框架）。所以使用了scrapy爬虫框架，scrapy框架是python开发的一个快速高层次抓取和web抓取框架，用于抓取web站点并从页面中提取结构化的数据，对于反爬虫很一般的网站而言，用scrapy完成的代码复用率很高，并且通过写middleware方便写一些统一的过滤器并通过管道方式存入数据库。我们可以进行简单的配置，即可对界面进行处理（如禁止渲染js，禁止加载图片）。

Scrapy是一个轻便的爬虫框架，具有很好的鲁棒性。我们更看重scrapy-redis组件，这是个分布式的爬虫框架，使用redis数据库替代scrapy默认的请求队列，并更换相应的操作及队列组件来达到多个爬虫执行一个任务加快爬取速度，实现了爬虫的分布式。原理如下：1.spider解析下载器下载下来的response,返回item或者是links

2.item或者links经过spidermiddleware的process\_spider\_out()方法，交给engine。

3.engine将item交给itempipeline,将links交给调度器

4.在调度器中，先将request对象利用scrapy内置的指纹函数，生成一个指纹对象

5.如果request对象中的dont\_filter参数设置为False,并且该request对象的指纹不在信息指纹的队列中，那么就把该request对象放到优先级的队列中

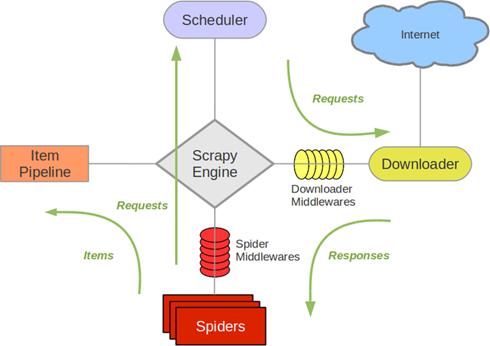
6.从优先级队列中获取request对象，交给engine

7.engine将request对象交给下载器下载，期间会通过downloadmiddleware的process\_request()方法

8.下载器完成下载，获得response对象，将该对象交给engine,期间会通过downloadmiddleware的process\_response()方法

9.engine将获得的response对象交给spider进行解析，期间会经过spidermiddleware的process\_spider\_input()方法

10.从第一步开始循环



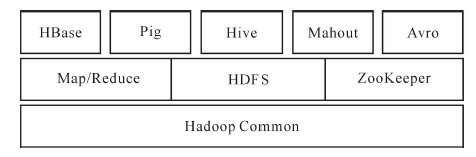
另外，我们计划更改内部的redis实现，使用我们自己的数据结构，来控制爬取过程。

另外我们还调研了apache lucene框架与apache nutch框架（介绍 与scrapy对比），最终我们选择了scrapy-redis，原因为scrapy为python语言，简单方便，scrapy-redis为现成的分布式框架，网络上资料较多。

分布式框架设计：

经过调研，我们决定基本使用hadoop生态圈的组件

Hadoop是一个分布式系统基础架构，由Apache基金会开发。用户可以在不了解分布式底层细节的情况下，开发分布式程序。充分利用集群的威力高速运算和存储。简单地说来，Hadoop是一个可以更容易开发和运行处理大规模数据的软件平台。



Common：一组分布式文件系统和通用I/O的组件与接口（序列化、Java RPC和持久化数据结构）。

MapReduce：分布式数据处理模型和执行环境，运行于大型商用机集群。

HDFS：分布式文件系统，运行于大型商用机集群。

Zookeeper：一个分布式、可用性高的协调服务，提供分布式锁之类的基本服务用于构建分布式应用。

HBase：一个分布式、按列存储数据库，使用HDFS作为底层存储，同时支持MapReduce的批量式计算和点查询（随机读取）。

Pig：一种数据流语言和运行环境，用以检索非常大的数据集，运行在MapReduce和HDFS的集群上。

Hive：一个分布式、按列存储的数据仓库，管理HDFS中存储的数据，并提供基于SQL的查询语言（由运行时引起翻译成MapReduce作业）用以查询数据。

Mahout：一个在Hadoop上运行的可扩展的机器学习和数据挖掘类库（例如分类和聚类算法）。

Avro：一种支持高效、跨语言的RPC以及永久存储数据的序列化系统。

Sqoop：在数据库和HDFS之间高效传输数据的工具。



另外我们也调研了spark生态圈



Spark生态圈也称为BDAS（伯克利数据分析栈），是伯克利APMLab实验室打造的，力图在算法（Algorithms）、机器（Machines）、人（People）之间通过大规模集成来展现大数据应用的一个平台。伯克利AMPLab运用大数据、云计算、通信等各种资源以及各种灵活的技术方案，对海量不透明的数据进行甄别并转化为有用的信息，以供人们更好的理解世界。该生态圈已经涉及到机器学习、数据挖掘、数据库、信息检索、自然语言处理和语音识别等多个领域。

Spark生态圈以Spark Core为核心，从HDFS、Amazon S3和HBase等持久层读取数据，以MESS、YARN和自身携带的Standalone为资源管理器调度Job完成Spark应用程序的计算。 这些应用程序可以来自于不同的组件，如Spark Shell/Spark Submit的批处理、Spark Streaming的实时处理应用、Spark SQL的即席查询、BlinkDB的权衡查询、MLlib/MLbase的机器学习、GraphX的图处理和SparkR的数学计算等等。

Spark内核架构：

1提供了有向无环图（DAG）的分布式并行计算框架，并提供Cache机制来支持多次迭代计算或者数据共享，大大减少迭代计算之间读取数据局的开销，这对于需要进行多次迭代的数据挖掘和分析性能有很大提升

2在Spark中引入了RDD (Resilient Distributed Dataset) 的抽象，它是分布在一组节点中的只读对象集合，这些集合是弹性的，如果数据集一部分丢失，则可以根据“血统”对它们进行重建，保证了数据的高容错性；

3移动计算而非移动数据，RDD Partition可以就近读取分布式文件系统中的数据块到各个节点内存中进行计算

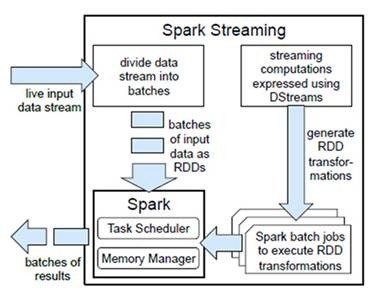
4使用多线程池模型来减少task启动开稍

5采用容错的、高可伸缩性的akka作为通讯框架

Spark streaming:

SparkStreaming是一个对实时数据流进行高通量、容错处理的流式处理系统，可以对多种数据源（如Kdfka、Flume、Twitter、Zero和TCP 套接字）进行类似Map、Reduce和Join等复杂操作，并将结果保存到外部文件系统、数据库或应用到实时仪表盘。

用一张图概括：



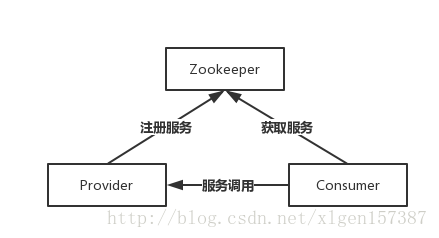
具有容错性，实时性，扩展性与吞吐量等特点。

最后我们可能会将此框架移植到apache spark上。

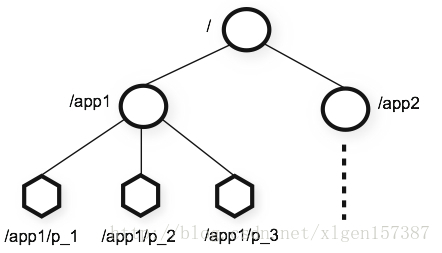
作为整个集群的协调与控制，我们使用apache zookeeper

ZooKeeper 是Hadoop下的一个子项目，它是一个针对大型分布式系统的可靠协调系统；它提供的功能包括：配置维护、名字服务、分布式同步、组服务等； 它的目标就是封装好复杂易出错的关键服务，将简单易用的接口和性能高效、功能稳定的系统提供给用户。

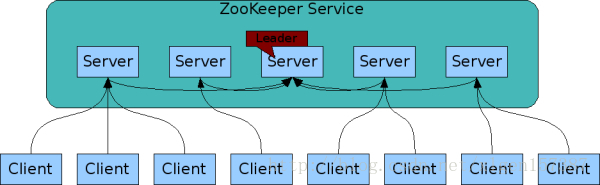
Zookeeper一个最常用的使用场景就是用于担任服务生产者和服务消费者的注册中心，服务生产者将自己提供的服务注册到Zookeeper中心，服务的消费者在进行服务调用的时候先到Zookeeper中查找服务，获取到服务生产者的详细信息之后，再去调用服务生产者的内容与数据，如图：



Zookeeper层次结构命名空间：



1、ZooKeeper 系统架构



ZooKeeper 的架构图中我们需要了解和掌握的主要有：

（1）ZooKeeper分为服务器端（Server） 和客户端（Client），客户端可以连接到整个 ZooKeeper服务的任意服务器上（除非 leaderServes 参数被显式设置， leader 不允许接受客户端连接）。

（2）客户端使用并维护一个 TCP 连接，通过这个连接发送请求、接受响应、获取观察的事件以及发送心跳。如果这个 TCP 连接中断，客户端将自动尝试连接到另外的 ZooKeeper服务器。客户端第一次连接到 ZooKeeper服务时，接受这个连接的 ZooKeeper服务器会为这个客户端建立一个会话。当这个客户端连接到另外的服务器时，这个会话会被新的服务器重新建立。

（3）上图中每一个Server代表一个安装Zookeeper服务的机器，即是整个提供Zookeeper服务的集群（或者是由伪集群组成）；

（4）组成ZooKeeper服务的服务器必须彼此了解。 它们维护一个内存中的状态图像，以及持久存储中的事务日志和快照， 只要大多数服务器可用，ZooKeeper服务就可用；

（5）ZooKeeper 启动时，将从实例中选举一个 leader，Leader 负责处理数据更新等操作，一个更新操作成功的标志是当且仅当大多数Server在内存中成功修改数据。每个Server 在内存中存储了一份数据。

（6）Zookeeper是可以集群复制的，集群间通过Zab协议（Zookeeper Atomic Broadcast）来保持数据的一致性；

（7）Zab协议包含两个阶段：leader election阶段和Atomic Brodcast阶段。

(a) 集群中将选举出一个leader，其他的机器则称为follower，所有的写操作都被传送给leader，并通过brodcast将所有的更新告诉给follower。

(b) 当leader崩溃或者leader失去大多数的follower时，需要重新选举出一个新的leader，让所有的服务器都恢复到一个正确的状态。

(c) 当leader被选举出来，且大多数服务器完成了 和leader的状态同步后，leadder election 的过程就结束了，就将会进入到Atomic brodcast的过程。

(d) Atomic Brodcast同步leader和follower之间的信息，保证leader和follower具有形同的系统状态。

Zookeeper特性：

(1)最终一致性：为客户端展示同一视图，这是 ZooKeeper 最重要的性能。

(2)可靠性：如果消息被一台服务器接受，那么它将被所有的服务器接受。

(3)实时性：ZooKeeper 不能保证两个客户端同时得到刚更新的数据，如果需要最新数据，应该在读数据之前调用sync()接口。

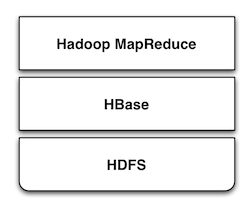
(4)等待无关（wait-free）：慢的或者失效的 client 不干预快速的client的请求。

(5)原子性：更新只能成功或者失败，没有中间其它状态。

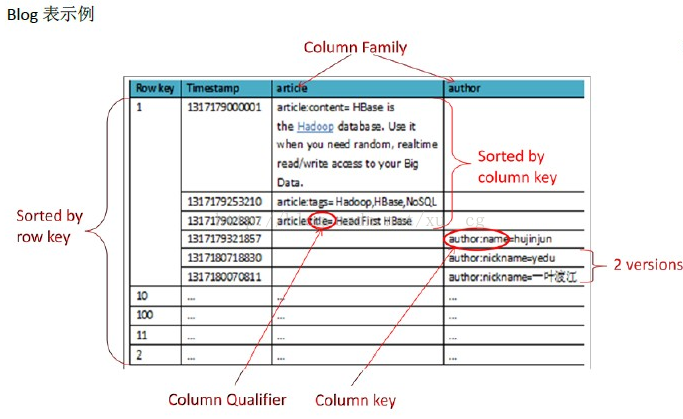
(6)顺序性：对于所有Server，同一消息发布顺序一致。

首先持久化存储使用apache hbase，

HBase是Apache Hadoop中的一个子项目，Hbase依托于Hadoop的HDFS作为最基本存储基础单元，通过使用hadoop的DFS工具就可以看到这些这些数据存储文件夹的结构,还可以通过Map/Reduce的框架(算法)对HBase进行操作：



基本概念：



Table（表）：HBase table 由多个 row 组成。

Row（行）：每一 row 代表着一个数据对象，每一 row 都是以一个 row key（行键）和一个或者多个 column 组成。row key 是每个数据对象的唯一标识的，按字母顺序排序，即 row 也是按照这个顺序来进行存储的。所以，row key 的设计相当重要，一个重要的原则是，相关的 row 要存储在接近的位置。比如网站的域名，row key 就是域名，在设计时要将域名反转（例如，org.apache.www、org.apache.mail、org.apache.jira），这样的话， Apache 相关的域名在 table 中存储的位置就会非常接近的。

Column（列）：column 由 column family 和 column qualifier 组成，由冒号（:）进行进行间隔。比如family:qualifier。

Column Family（列族）：在 HBase，column family 是 一些 column 的集合。一个 column family 所有 column 成员是有着相同的前缀。比如， courses:history 和 courses:math 都是 courses 的成员。冒号（:）是 column family 的分隔符，用来区分前缀和列名。column 前缀必须是可打印的字符，剩下的部分列名可以是任意字节数组。column family 必须在 table 建立的时候声明。column 随时可以新建。在物理上，一个的 column family 成员在文件系统上都是存储在一起。因为存储优化都是针对 column family 级别的，这就意味着，一个 column family 的所有成员的是用相同的方式访问的。

Column Qualifier（列限定符）：column family 中的数据通过 column qualifier 来进行映射。column qualifier 也没有特定的数据类型，以二进制字节来存储。比如某个 column family “content”，其 column qualifier 可以设置为 “content:html” 和 “content:pdf”。虽然 column family 是在 table 创建时就固定了，但 column qualifier 是可变的，可能在不同的 row 之间有很大不同。

Cell（单元格）：cell 是 row、column family 和 column qualifier 的组合，包含了一个值和一个 timestamp，用于标识值的版本。

Timestamp（时间戳）：每个值都会有一个 timestamp，作为该值特定版本的标识符。默认情况下，timestamp 代表了当数据被写入 RegionServer 的时间，但你也可以在把数据放到 cell 时指定不同的 timestamp。

Hbase应用场景：

1.半结构化或非结构化数据

对于数据结构字段不够确定或杂乱无章很难按一个概念去进行抽取的数据适合用HBase。以上面的例子为例，当业务发展需要存储author的email，phone，address信息时RDBMS(i.e. 关系数据库管理系统(Relational Database Management System))需要停机维护，而HBase支持动态增加.

2.记录非常稀疏

RDBMS的行有多少列是固定的，为null的列浪费了存储空间。HBase为null的Column不会被存储，这样既节省了空间又提高了读性能。

3.多版本数据

如根据Row key和Column key定位到的Value可以有任意数量的版本值，因此对于需要存储变动历史记录的数据，用HBase就非常方便了。

4.超大数据量

当数据量越来越大，RDBMS数据库撑不住了，就出现了读写分离策略，通过一个Master专门负责写操作，多个Slave负责读操作，服务器成本倍增。随着压力增加，Master撑不住了，这时就要分库了，把关联不大的数据分开部署，一些join查询不能用了，需要借助中间层。随着数据量的进一步增加，一个表的记录越来越大，查询就变得很慢，于是又得搞分表，比如按ID取模分成多个表以减少单个表的记录数。经历过这些事的人都知道过程是多么的折腾。采用HBase就简单了，只需要加机器即可，HBase会自动水平切分扩展，跟Hadoop的无缝集成保障了其数据可靠性（HDFS）和海量数据分析的高性能（MapReduce）。

Hbase优点：

列的可以动态增加，并且列为空就不存储数据，节省存储空间

Hbase 自动切分数据，使得数据存储自动具有水平扩展

Hbase 可以提供高并发读写操作的支持

与 Hadoop MapReduce 相结合有利于数据分析

容错性

版权免费

非常灵活的模式设计（或者说没有固定模式的限制）

可以跟 Hive 集成，使用类 SQL 查询

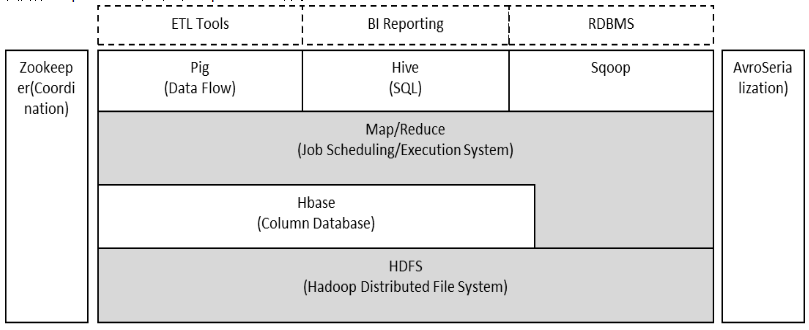
自动故障转移

客户端接口易于使用

行级别原子性，即，PUT 操作一定是完全成功或者完全失败

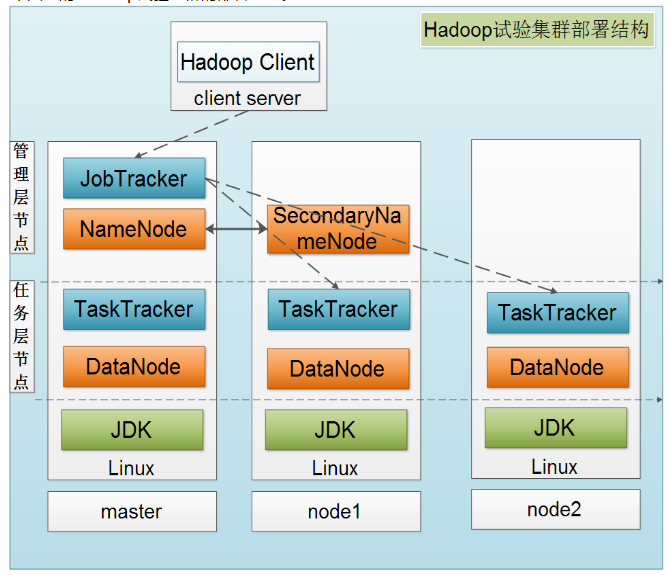
经过我们实验，可以只使用hbase而不是用hadoop组件，但是为了以后的可扩展性考虑，我们还是使用了apache hadoop组件

Hadoop由HDFS、MapReduce、HBase、Hive和ZooKeeper等成员组成，其中最基础最重要元素为底层用于存储集群中所有存储节点文件的文件系统HDFS（Hadoop Distributed File System）来执行MapReduce程序的MapReduce引擎。



（1）Pig是一个基于Hadoop的大规模数据分析平台，Pig为复杂的海量数据并行计算提供了一个简单的操作和编程接口；   
（2）Hive是基于Hadoop的一个工具，提供完整的SQL查询，可以将sql语句转换为MapReduce任务进行运行；   
（3）ZooKeeper:高效的，可拓展的协调系统，存储和协调关键共享状态；   
（4）HBase是一个开源的，基于列存储模型的分布式数据库；   
（5）HDFS是一个分布式文件系统，有着高容错性的特点，适合那些超大数据集的应用程序；   
（6）MapReduce是一种编程模型，用于大规模数据集（大于1TB）的并行运算。

典型的Hadoop集群的部署结构：



Hadoop特性：

分布 - 而不是构建一个大的超级计算机，存储和处理分散在一组较小的机器，通信和协同工作

水平可扩展性 - 只需添加新机器即可轻松扩展Hadoop集群。每个新机器增加了Hadoop集群的总存储和处理能力

容错 - 即使少数硬件或软件组件无法正常工作，Hadoop仍可继续运行。成本优化 - Hadoop在标准硬件上运行;它不需要昂贵的服务器

编程抽象 - Hadoop负责处理与分布式计算相关的所有混乱细节。由于高级API，用户可以专注于实现解决他们现实世界问题的业务逻辑

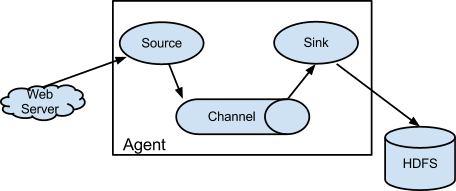
数据位置 - 不需要将大数据集移动到应用程序正在运行的位置，而是运行数据已经在的应用程序

我们参考了美团的日志收集系统（<https://tech.meituan.com/mt-log-system-arch.html?from=singlemessage>），

首先我们决定使用apache flume，版本为flume-ng，

Flume是Apache下面的一个分布式组件，它提供高效，可靠的收集，整合，传输日志数据的服务。Flume可以理解成一个管道，它连接数据的生产者和消费者，它从数据的生产者(Source)获取数据，保存在自己的缓存(Channel)中，然后通过Sink发送到消费者。它不对数据做保存和复杂的处理(可以做简单过滤和改写)。

由于Flume-og版本中Flume的角色过多，比如agent, collector,master等，导致用户使用起来存在困难，因此新的Flume-ng只设计了一个角色Agent：



Flume的Agent有如下特点：

1.一个Agent是一个单独的进程。

2.一个Agent由一个或多个管道组成。

3.一个管道由三个内部模块组成Source,Channel,Sink。

4.用户可以随意指定Agent中Source，Channel，Sink的类型，只要该类型符合用户自己的使用场景，Flume提供了相当大的灵活度，用户可以随意搭配。

Source

Source是Flume里面的数据生成模块，它一般有三种方式：   
1. Source作为服务的，开启端口，接收其他服务发来的数据，如Avro，NetCat。   
2. Source作为客户端，从其他服务获取数据，如Kafka。   
3. Source自动生成数据，一般做调试/测试使用，如Seq，StressSource。

Flume内置提供了很多开箱即用的Source供用户选择，包括Avro,Kafka,NetCat,   
Exec,SyslogTcp,StressSource等。另外，Flume也支持用户自定义Source。

Channel

Channel是Flume里面的数据缓存模块，它只提供数据的临时缓存。   
Source将收到的数据放到Channel中，待Sink从Channel中取走该数据后,   
Channel将清除该数据。

Flume内置了一些Channel供用户选择使用，比如Memory, File, Kafka, JDBC。   
大体上分为内存，文件，外部存储系统，其中内存方式是最快速的。当然，用户也可以定制化自己的Channel。

Sink

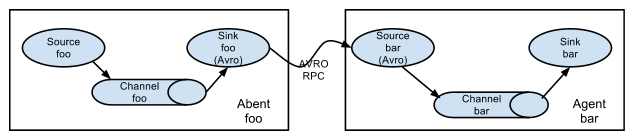
Sink是Flume里面的数据处理模块，它负责从Channel中消费数据，可以存到HDFS，HBASE等数据库中，也有可以发给下一级Agent。

Flume内置了一些Channel供用户选择使用，比如Kafka，HBase, AsyncHBase,Logger，null等，当然，用户也可以定制化自己的Sink。

Flume的架构设计十分灵活，依据用户的需求自行搭建，总的来说有如下一些使用方式

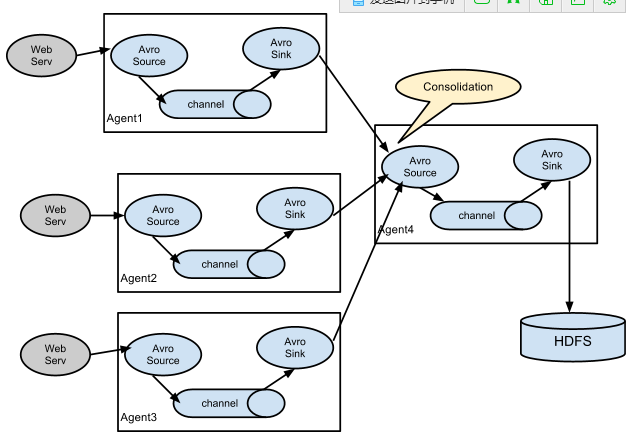
级联

在某些场景下，我们需要将两个agent级联起来。如果一个agent的Source为选为Avro类型，而另一个agent的Sink也选为Avro类型，那么我们可以将两个agent级联起来，只需要将下一级agent的IP信息配置到上一级的Sink配置中即可。

(当然，是由于内置的Avro的Sink和内置的Avro类型的Source能对接起来，才能完成级联，如果用户有其他自定义的Source和Sink也能完成对接，那么也可以会用自定义的类型。)

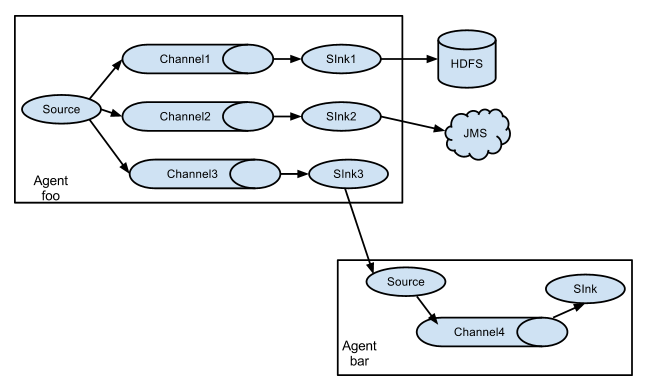
聚合

在一些场景下，我们需要收集多个系统的日志聚合到一个地方，因此需要将agent合并到一起。我们只需要将多个agent的sink指向一个agent即可。



分发

在某些场景下，我们需要将一个系统的日志收集后，f存到多个目的地，Flume支持这种应用。事实上，从一个source出来的数据，我们可以选择数据是简单的复制分发到所有的channel,，还是有针对数据做选择性的分发，比如图片发送到Channel A， 文本发送到Channel B等类似的功能。



总的来说，flume-ng有如下一些优点：

1.大大降低了对用户的要求，如不再依赖 zookeeper，用户无需去搭建 zookeeper 集群

2.用户也不再纠结于 OG 中的模糊概念（尤其是 physical nodes、logical nodes，agent、collector）

3.有利于 Flume 和其他技术、hadoop 周边组件的整合，比如在 NG 版本中，Flume 轻松实现了和 jdbc、hbase 的集成

4.将 OG 版本中复杂、大规模、不稳定的标签移除，Flume 实现了向灵活、轻便的转变，而且在功能上更加强大、可扩展性更高

然后实时计算部分使用apache storm（原本为twitter的组件，不是linkedin的storm），

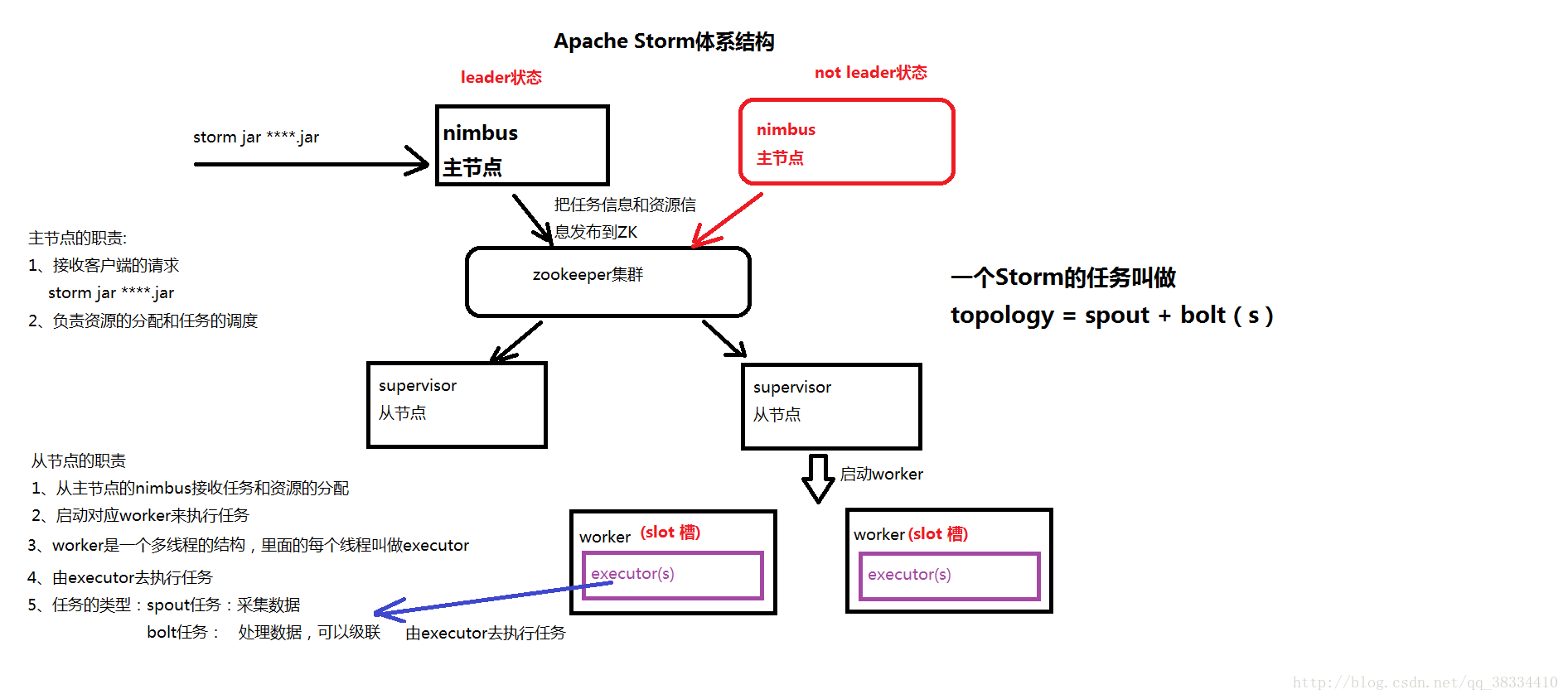
Storm为分布式实时计算提供了一组通用原语,可被用于”流处理”之中,实时处理消息并更新数据库.这是管理列队及工作者集群的另一种方式.Storm也可以被用于”连续计算”,对数据结构做连续查询, 在计算时就将结果以流的形式输出给用户.还可以被用于”分布式RPC”,以并行的方式运行昂贵的运算.   
Strom可以方便在一个计算集群中编写与扩展复杂的实时就算,storm用于实时处理,就好比Hadoop用于批量处理.Storm保证每个消息都会得到处理,而且很快的在一个小集群中,每秒可以处理数以百万计的消息.更好的是你可以使用任意编程语言来开发.

Storm适用的场景：

1、流数据处理：Storm可以用来用来处理源源不断的消息，并将处理之后的结果保存到持久化介质中。

2、分布式RPC：由于Storm的处理组件都是分布式的，而且处理延迟都极低，所以可以Storm可以做为一个通用的分布式RPC框架来使用。

Storm的体系结构：



1.Nimbus 负责在集群里面发送代码，分配工作给机器，并且监控状态。全局只有一个。   
2.Supervisor 会监听分配给它那台机器的工作，根据需要启动/关闭工作进程Worker。每一个要运行Storm的机器上都要部署一个，并且，按照机器的配置设定上面分配的槽位数。   
3.Zookeeper是Storm重点依赖的外部资源。Nimbus和Supervisor甚至实际运行的Worker都是把心跳保存在Zookeeper上的。Nimbus也是根据Zookeerper上的心跳和任务运行状况，进行调度和任务分配的。   
4.Topology 处理的最小的消息单位是一个Tuple，也就是一个任意对象的数组。Storm提交运行的程序称为Topology。   
5.Topology 由Spout和Bolt构成。Spout是发出Tuple的结点。Bolt可以随意订阅某个Spout或者Bolt发出的Tuple。Spout和Bolt都统称为component。

另外我们调研了spark streaming，

Spark Streaming 是Spark核心API的一个扩展，可以实现高吞吐量的、具备容错机制的实时流数据的处理。支持从多种数据源获取数据，包括Kafk、Flume、Twitter、ZeroMQ、Kinesis 以及TCP sockets，从数据源获取数据之后，可以使用诸如map、reduce、join和window等高级函数进行复杂算法的处理。最后还可以将处理结果存储到文件系统，数据库和现场仪表盘。在“One Stack rule them all”的基础上，还可以使用Spark的其他子框架，如集群学习、图计算等，对流数据进行处理。

Spark Streaming处理的数据流图：



Spark的各个子框架，都是基于核心Spark的，Spark Streaming在内部的处理机制是，接收实时流的数据，并根据一定的时间间隔拆分成一批批的数据，然后通过Spark Engine处理这些批数据，最终得到处理后的一批批结果数据。

对应的批数据，在Spark内核对应一个RDD实例，因此，对应流数据的DStream可以看成是一组RDDs，即RDD的一个序列。通俗点理解的话，在流数据分成一批一批后，通过一个先进先出的队列，然后 Spark Engine从该队列中依次取出一个个批数据，把批数据封装成一个RDD，然后进行处理，这是一个典型的生产者消费者模型，对应的就有生产者消费者模型的问题，即如何协调生产速率和消费速率。

Storm与Spark Streming比较：



最终决定使用storm。

我们发现flume与storm直接相连的实例很少，经过调研与参考，决定使用apache kafka，作为组件中间件，起到数据通道的作用。此后经过我们的实验，决定停止显式使用kafka，而是把它作为flume中的内嵌部分使用。

Kafka是一个基于分布式的消息push-subscribe系统，它被设计成快速、可扩展的、持久的。与其他消息发布-订阅系统类似，Kafka在主题当中保存消息的信息。生产者向主题写入数据，消费者从主题读取数据。Kafka设计中将topic的每个分区当作一个具有顺序的日志。同处于一个分区中的消息都被设置了一个唯一的偏移量。Kafka只会保持跟踪未读消息，一旦消息被置为已读状态，Kafka就不会再去管理它了。

Kafka是一个分布式数据流平台。

在系统或者应用之间建立实施的数据流管道，可靠的传输数据。建立实时的流式应用，用来传输数据流或者对数据流做出响应。kafka提供以下三个核心能力。

(1) 可以发布、订阅数据流中的记录。这一点比较类似于消息队列。

(2) 可以以容错的方式存储数据流。

(3) 可以及时处理数据流

2. Kafka特性

(1) 高吞吐量、低延迟kafka每秒可以处理几十万条消息，它的延迟最低只有几毫秒。

(2) 可扩展性kafka集群支持热扩展。

(3) 持久性、可靠性消息被持久化到本地磁盘，并且支持数据备份防止数据丢失。

(4) 容错性允许集群中节点失败（若副本数量为n，则允许n-1个节点失败）。

(5) 高并发支持数千个客户端同时读写。

二、  术语

1.  Broker

Kafka集群包含一个或多个服务器，这种服务器被称为broker。

2.  Topic

每条发布到Kafka集群的消息都有一个类别，这个类别被称为Topic。物理上不同Topic的消息分开存储，逻辑上一个Topic的消息虽然保存于一个或多个broker上，但用户只需指定消息的Topic即可生产或消费数据而不必关心数据存于何处。

3.  Partition

Partition是物理上的概念，每个Topic包含一个或多个Partition。

4.  Producer

负责发布消息到Kafka broker。

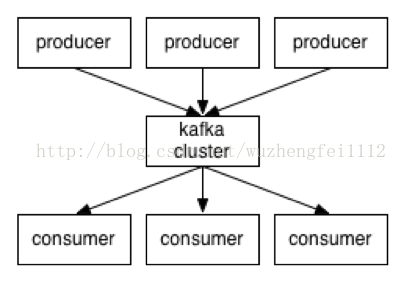
5.  Consumer

消息消费者，向Kafka broker读取消息的客户端。

6.  ConsumerGroup

各个consumer可以组成一个Group，每个消息只能被Group中的一个consumer消费，如果一个消息可以被多个consumer消费的话，那么这些consumer必须在不同的组。可为每个Consumer指定group name，若不指定group name则属于默认的group。

1.   producer-broker-consumer



2.   集群部署模型

