**Topics/logs**

Kafka和JMS(Java Message Service)实现(activeMQ)不同的是：即使消息被消费，消息仍然不会被立即删除，日志文件将会根据broker中的配置要求,保留一定的时间之后删除；比如log文件保留2天，那么两天后，文件会被清楚，无论其中的消息是否被消费。Kafka通过这种简单，来释放磁盘空间，以及减少消息消费之后对文件内容改动的磁盘IO开支。

对于consumer而言，它需要保存消费消息的offset，对于offset的保存和使用，有consumer来控制；当consumer正常消费消息时，offset将会”线程”的向前驱动，即消息将依次顺序被消费，实际上consumer可以使用任意顺序消费消息，它只需要将offset重置为任意值（offset将会保存在zookeeper中）。

Kafka集群几乎不需要维护任何consumer和producer状态信息，这些信息由zookeeper保存；因此producer和consumer的客户端实现非常轻量级，它们可以随意离开，而不会对集群造成额外的影响。

Partitions的设计目的有多个，最根本原因是kafka基于文件存储，通过分区，可以将日志内容分散到多个server上，来避免文件尺寸达到单机磁盘的上限，每个partiton都会被当前server（kafka实例）保存；可以将一个topic切分多任意多个partitions，来提高保存/消费的效率。此外越多的partitions意味着可以容纳更多的consumer，有效提升并发消费的能力。

**Distribution**

一个Topic的多个partitions，被分布在kafka集群中的多个server上；每个server（kafka实例）负责partitions中消息的读写操作；此外kafka还可以配置partitions需要备份的个数（replicas），每个partition将会被备份到多台机器上，以提高可用性。

基于replicated方法，那么就意味着需要对多个备份进行调度；每个partition都有一个server为”leader”；leader负责所有的读写操作，如果leader失效，那么将会有其他follower来接管（成为新的leader）；follower只是单调的和leader跟进，同步消息即可，由此可见作为leader的server承载了全部的请求压力，因此从集群的整体考虑，有多少个partitions就意味着有多少个”leader”，kafka会将”leader”均衡的分散在每个实例上,来确保整体的性能稳定。

Producers

Producer将消息发布到指定的Topic中，同时Producer也能决定将此消息 归属于哪个partition；比如基于”round-robin”方式或通过其他的一些算法等。

Consumer

本质上kafka只支持Topic，每个consumer属于一个consumer group；反过来说，每个group中可以有多个consumer发送到Topic的消息，只会被订阅此Topic的group中的一个consumer消费。

如果所有的consumer都具有相同的group，这种情况和queue模式很像；消息将会在consumers之间负载均衡。如果所有的consumer都具有不同的group，那这就是”发布-订阅”；消息将会广播给所有的消费者。在kafka中，一个partition中的消息只会被group中的一个consumer消费；每个group中consumer消息消费互相独立；我们可以认为一个group是一个”订阅者”，一个Topic中的每个partitions，只会被一个”订阅者”中的一个consumer消费,不过一个consumer可以消费多个partitions中的消息，kafka只能保证一个partition中的消息被某个consumer消费时，消息是顺序的，事实上，从Topic角度来说，消息仍不是有序的。

Kafka的设计原理决定，对于一个topic，同一个group中不能有多于partitions个数的consumer同时消费，否则将意味着某些consumer将无法得到消息。

Guarantees

1. 发送到partitions中的消息将会按照它接受的顺序追加到日志中。
2. 对于消费者而言，他们消费消息的顺序和日志中消息顺序一致。
3. 对于Topic的”replicationfactor”为N，那么允许N-1个kafka实例失效。

二． 使用场景

1. Messaging

对于一些常规的消息系统，kafka是个不错的选择；partitions/replication和容错，可以使kafka具有良好的扩展性和性能优势，不过到目前为止，我们应该很清楚意识到，kafka并没有提供JMS中的”事务性”消息传输担保(消息确认机制)、”消息分组”等企业特性；kafka只能使用作为”常规”的消息系统，在一定程度上，尚未确保信息的发送与接受绝对可靠(比如，消息重发，消息发送丢失等)。

2、Websit activity tracking

kafka可以作为"网站活性跟踪"的最佳工具;可以将网页/用户操作等信息发送到kafka中.并实时监控,或者离线统计分析等

3、Log Aggregation

Kafka的特性决定它非常适合作为”日志收集中心”;application可以将操作日志”批量、异步”的发送到kafka集群中，而不是保存在本地或者DB中;kafka可以批量提交消息/压缩消息等，这对producer端而言，几乎感觉不到性能的开支。此时consumer端可以是hadhoop等其他系统化存储和分析系统。

三． 设计原理

Kafka的设计初衷是希望作为一个统一的信息收集平台，能够实时的收集反馈信息，并需要能够支撑较大的数据量，且具备良好的容错能力。

1. 持久性

Kafka使用文件存储消息，这就直接决定kafka在性能上严重依赖系统文件本身，且无论任何OS下，对文件系统本身的优化几乎没有可能，文件缓存/直接内存映射等是常用的手段，因为kafka是对日志文件进行append操作，因此磁盘检索的开支是较小的；同时为了减少磁盘写入的次数，broker会将消息暂时buffer起来，当消息的个数达到阀值时，再flush到磁盘，这样减少了磁盘IO调用的次数。

1. 性能

需要考虑的影响性能点很多，除磁盘IO之外，我们还需要考虑IO，这直接关系到kafka的吞吐量问题，kafka并没有提供太多高超的技巧；对于producer端，可以将消息buffer起来，当消息的条数,当消息的条数达到一定阀值时,批量发送给broker；对于consumer端也是一样，批量fetch多条消息，不过消息量的大小可以通过配置文件来制定，对于kafka broker端，似乎有个sendfile系统调用可以潜在的提升网络IO的性能：将文件的数据映射到系统内存中，socket直接读取相应的内存区域即可, 而无需进程再次copy和交换。其实对于producer/consumer/broker三者而言,CPU的开支应该都不大,因此启用消息压缩机制是一个良好的策略;压缩需要消耗少量的CPU资源,不过对于kafka而言,网络IO更应该需要考虑.可以将任何在网络上传输的消息都经过压缩.kafka支持gzip/snappy等多种压缩方式。

1. 生产者

负载均衡: producer将会和Topic下所有partition leader保持socket连接;消息由producer直接通过socket发送到broker，中间不会经过任何"路由层".事实上,消息被路由到哪个partition上,有producer客户端决定，比如可以采用random、key-hash、轮询等，如果一个Topics中有多个partition，那么在producer端实现”消息均衡分发”是必要的。

其中partition leader的位置(host:port)注册在zookeeper中, producer作为zookeeper client,已经注册了watch用来监听partition leader的变更事件.

异步发送：将多条消息暂且在客户端buffer起来，并将他们批量的发送到broker，小数据IO太多，会拖慢整体的网络延迟，批量延迟发送事实上提升了网络效率。不过这也有一定的隐患，比如说当producer失效时，那些尚未发送的消息将会丢失。

1. 消费者

Consumer端向broker发送fetch请求，并告知其获取信息的offset；此后consumer将会获得一定条数的消息；consumer端也可以重置offset来重新消费消息。

在JMS实现中,Topic模型基于push方式,即broker将消息推送给consumer端，不过在kafka中，采用了pull方式，即consumer和broker建立连接后，主动去pull或fetch消息；这种模式有些有点，consumer可以根据自己的消费能力适时的去fetch消息并处理，且可以控制消息消费的进度(offset)；此外，消费者可以良好的控制消息一次消费的数量，batch fetch。

其他JMS实现,消息消费的位置是有prodiver保留,以便避免重复发送消息或者将没有消费成功的消息重发等, 同时还要控制消息的状态.这就要求JMS broker需要太多额外的工作。在kafka中，partition中的消息只有一个consumer在消费，不存在消息状态的控制，也没有复杂的消息确认机制。可见kafka broker端是相当轻量级的。当消息被consumer接收后，consumer可以在本地保存最后消息的offset，并间歇性的向zookeeper注册offset，所以consumer客户端也是轻量级的。

1. 消息传送机制

对于JMS实现，消息传输担保非常直接：有且只有一次，在kafka中稍有不同。

1. at most once : 最多一次，这个和JMS中”非持久化”消息类似，发送一次,无论成败，将不会重发。

消费者fetch消息，然后保存offset，然后处理信息；当client保存offset之后，但在消息处理过程中出现了异常，导致部分消息未能继续处理。那么此后”未处理”的消息将不能被fetch到。

1. at least once: 消息至少发送一次,如果消息未能接受成功,可能会重发,直到接收成功。

消费者fetch信息，然后处理消息，然后保存offset，如果消息处理成功之后，但是在保存offset阶段zookeeper异常导致保存操作未能执行成功，这就导致接下来再次fetch时可能获得上次已经处理过的消息，这就导致接下来再次fetch时可能获得上次已经处理过的消息,这就是"at least once"，原因offset没有及时的提交给zookeeper，zookeeper恢复正常还是之前offset状态.

1. exactly once: 消息只会发送一次.
2. 复制备份
3. 日志
4. 分配