Flink使用（二）——Flink集群资源规划

目录

前言

本文主要译自Flink Forward 2017的柏林站中*Robert Metzger*的有关集群规划的How to size your flink cluster一文。该文中主要是考虑网络资源，博主结合自己的使用经验对文中省略的做了一定补充，同时也非常欢迎大伙留言补充。

本文非直译，原文链接如下：https://www.ververica.com/blog/how-to-size-your-apache-flink-cluster-general-guidelines

文中拿捏不准的地方，均附有英文原文。若是有表述不合适的，欢迎大伙留言指出。

回到顶部

1、关键参数与资源

为估算Flink集群所需资源，首先我们需要根据Flink任务中的指标给出集群的最低资源需求（baseline）。

1.1 指标（metric）：

1）每秒的record数和每个record的大小；

2）不同key的个数和每个key产生state的大小；

3）state的更新方式以及state的访问模式

此外还需考虑SLA（服务级别协议）。例如，可能愿意接受的停机时间，可接受的延迟或最大吞吐量，因为此类SLA会对Flink群集的大小产生影响。

1.2 资源

在给Flink集群做规划时，我们需要考虑集群的资源，但这里的资源一般指什么呢？一般有以下几种：

1）网络容量。在考虑网络容量时，我们也需要考虑到可能使用网络的其他服务，如Kafka、HDFS等；

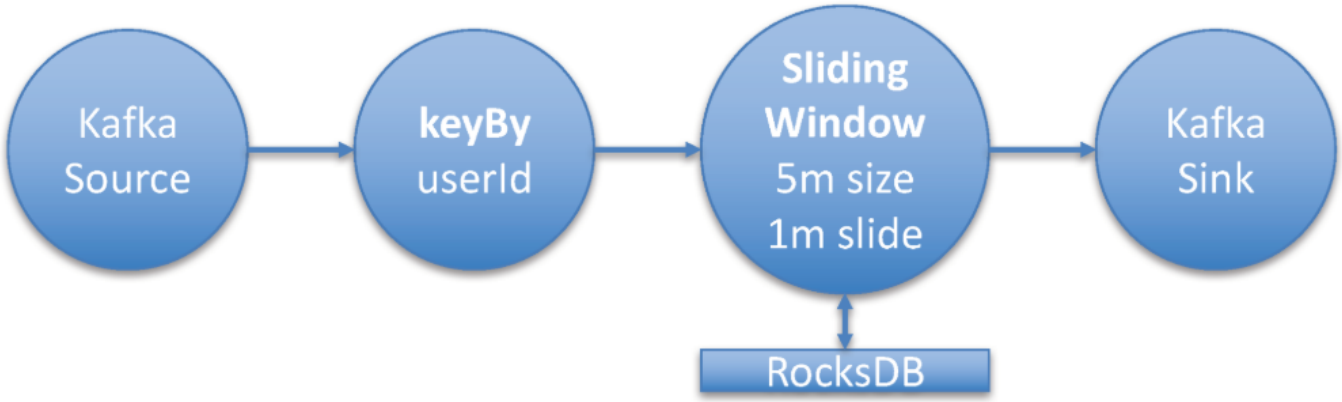
2）磁盘带宽。当我们的容错机制是基于磁盘的，如RockDB、HDFS，此时也有可能需要考虑到Kafka，因为其也是将数据存在磁盘的；

3）节点数量以及能提供的CPU和内存；

回到顶部

2、例子

Flink例子的拓扑图1如下：



该例子从kafka消费message，以用户id（userId）做keyBy后，经过window算子聚合（window算子为sliding window，其窗口大小为5min，间隔是1min），处理后的消息写入到kafka中。

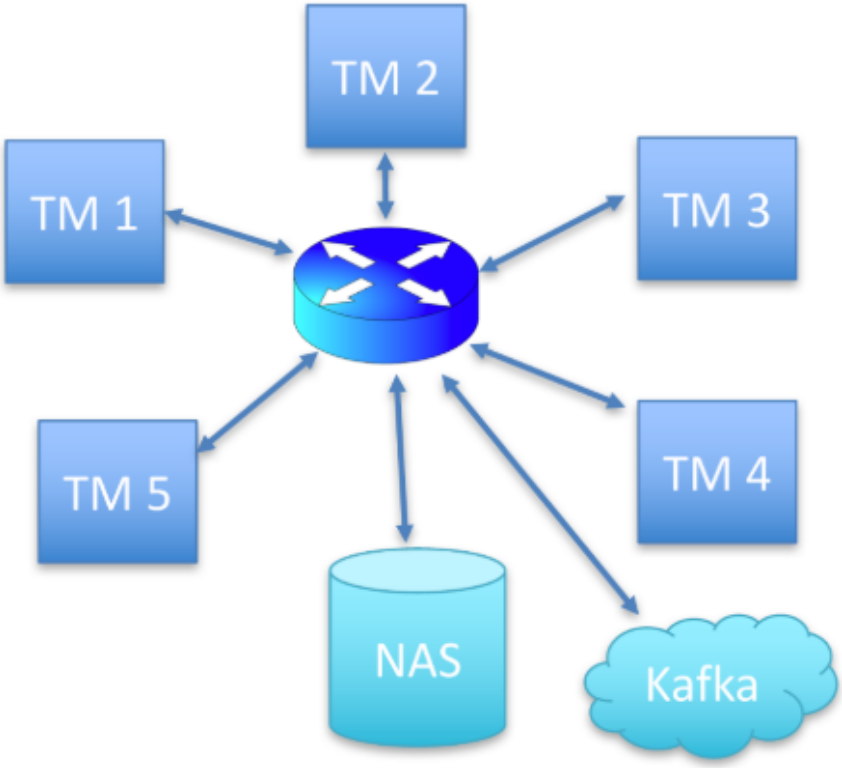
2.1 任务metrics

从kafka消费的record平均大小为2KB，吞吐量为1百万/s，userId的个数为5亿(5*10^9）。该任务的关键指标（key metric）如下：

- **Message size**: 2KB
- **Throughput**: 1,000,000 msg/sec
- **Distinct keys**: 500,000,000 (aggregation in window: 4 longs per key)
- **Checkpointing**: Once every minute.

2.2 硬件

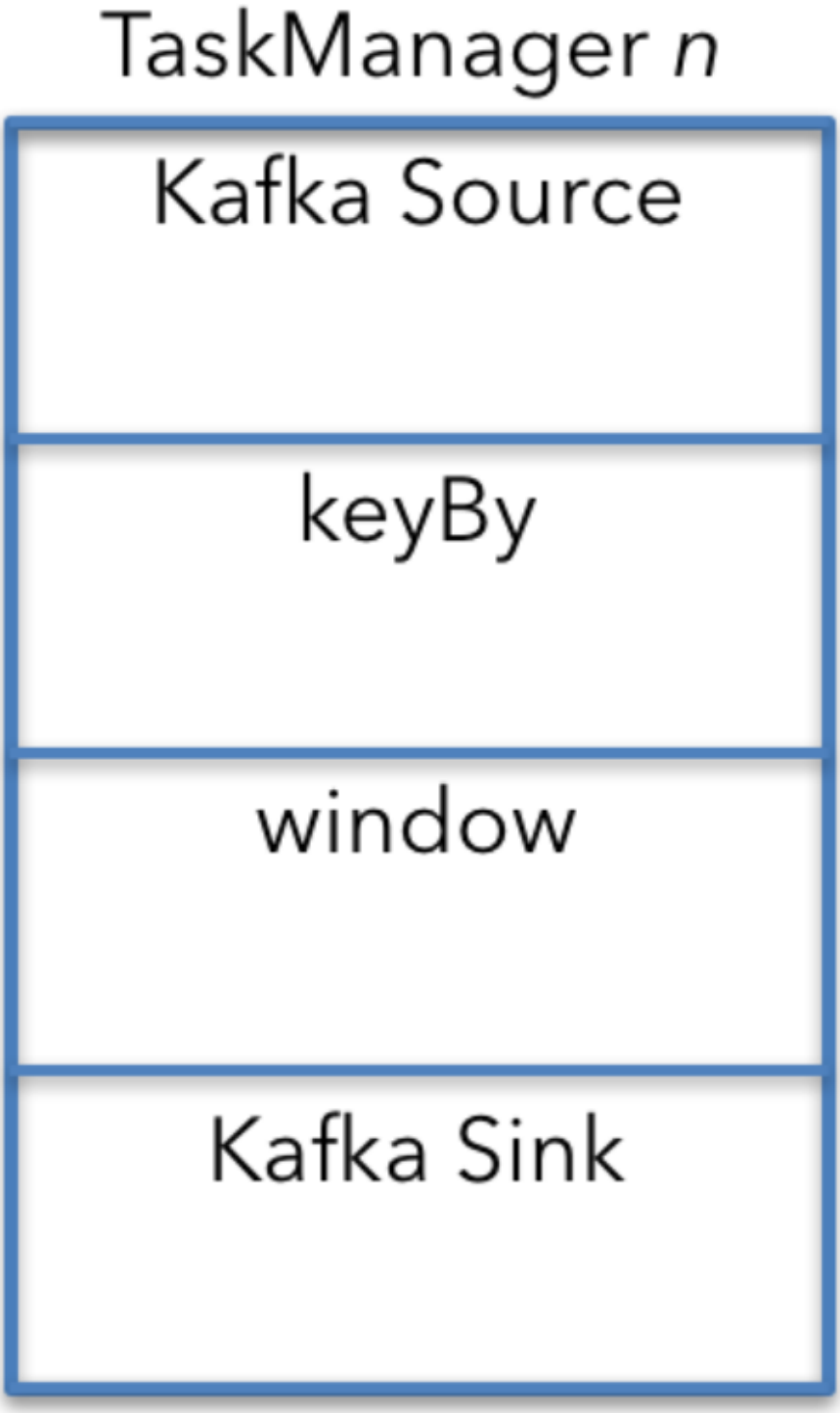
1）5个节点，每个节点有一个TaskManager；2）万兆网；3）磁盘通过网络连接（本例中集群部署在云上，物理机得另外考虑）；此外，kafka是单独的集群。如下图2：



每个节点是16核，为简化，文中暂不考虑CUP和内存的需求。在实际的生产中需要根据任务逻辑和容错方式去考虑内存。本例的状态是通过RockDB的方式存储，该方式对内存的要求较小。

2.3 单节点资源需求

为方便分析，我们先考虑单节点上的资源需求，集群整体的需求可以大致通过乘以节点数得到。例子中，每个算子的并行度相同且没有其他特殊调度限制，每个节点拥有流任务的所有算子，即每个节点上都有Kafka source、window、Kafka sink算子，如下图3：



为方便计算资源，上图中KeyBy算子单独给出，但在实际中KeyBy是Kafka算子和window算子之间链接的配置属性。下面将结合图3从上往下分析网络资源的需求（network resource requirement）。

2.3.1 Kafka Source

为计算从单个Kafka Source的拿到的数据，我们先计算从Kafka拿到数据的综合，计算过程如下：

1）每秒1,000,000条，每条大小为2KB ，每秒获得总数据为：

2KB×1,000,000/s=2GB/s

2）Flink集群中每个节点每秒获得数据为

2GB/s÷5=400MB/s

2.3.2 Shuffle过程（KeyBy）

经过KeyBy后，具有相同userId的数据将会在一个节点上，但是Kafka可能根据不同的元数据进行分区（partitioned according to a different partitioning scheme），因此对一个key（userId），单个节点直接从Kafka得到的数据为400MB/s÷5=80MB/s，这样就有320MB/s的需要通过shuffle获得。

2.3.3 window emit和Kafka sink

window会发送多少数据，有多少数据会到Kafka sink？分析如下：

window算子为每个key（userId）聚合生成4个long数，每分钟发射一次，这样window每分钟为每个key会发射2个int字段（userId、window_ts）和4个long字段，总的数据量如下：

(2 x 4 bytes) + (4 x 8 bytes) = 40 bytes per key

这样5个节点，每个节点的数据量为：

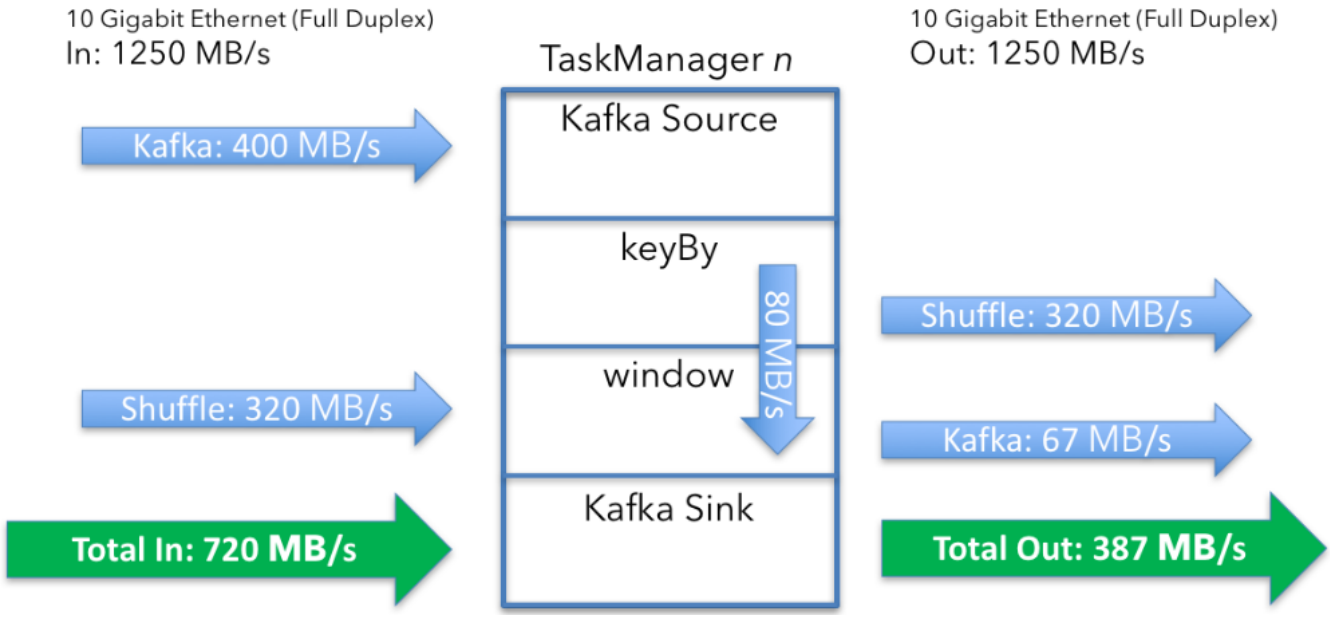
500,000,000 keys x 40 bytes÷5 = 4GB

每秒的数据量为4GB/min ÷ 60 = 67MB/s，因为每个节点上都有Kafka sink，不需要额外的重分区，因此从Flink到Kafka的数据为67MB/s。在实际中，算子不会以67MB/s的恒定速度发送数据，而是每分钟最大限度地利用可用带宽几秒钟。

单节点数据总流向总结如下：

- **Data in**: 720MB/s (400 + 320) per machine
- **Data out**: 387MB/s (320 + 67) per machine

整个过程可以总结如下：

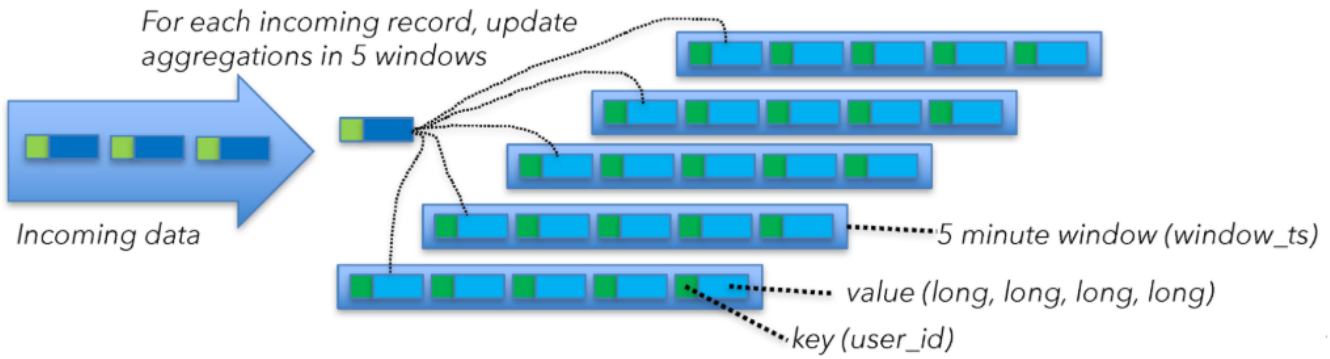


2.3.4 获取state和checkpointing

到目前为止，我们只考虑Flink处理的数据。实际上，还需考虑到state存储和checkpoint过程中所需要的网络资源。

1）state消耗的网络带宽

为弄清window算子的state大小，我们需要从另外一个角度去分析该问题。Flink的计算窗口大小为5min，滑动尺度为1min，为此Flink通过维持五个窗口实现"滑动窗口"。如在2.3.3节中提到，每个窗口Flink需要维持40Bytes的数据。每当一个event到达时，Flink将会从已有state中获得数据（40Bytes）去更新聚合值，然后将更新后的数据写入state（磁盘），如下图：

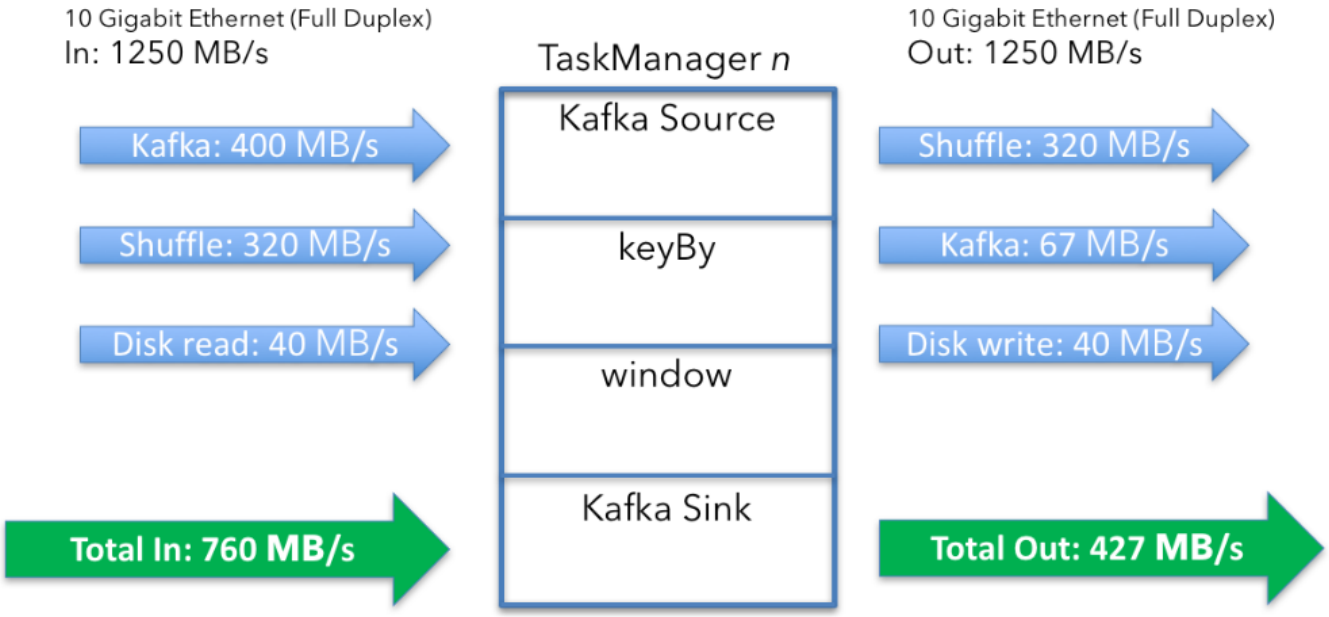


这意味每个节点将会产生40MB/s的网络消耗，计算方式如下：

40 bytes of state x 5 windows x 200,000 msg/s per machine = 40MB/s

正如文中开始提及的，磁盘是通过网络连接的，所以state读取产生的网络消耗也得考虑进去，则单节点整体的网络资源情况如下：

- **Data in**: 760MB/s (400 MB/s data in + 320 MB/s shuffle + 40 MB/s state)
- **Data out**: 427MB/s (320 MB/s shuffle + 67 MB/s data out + 40 MB/s state)



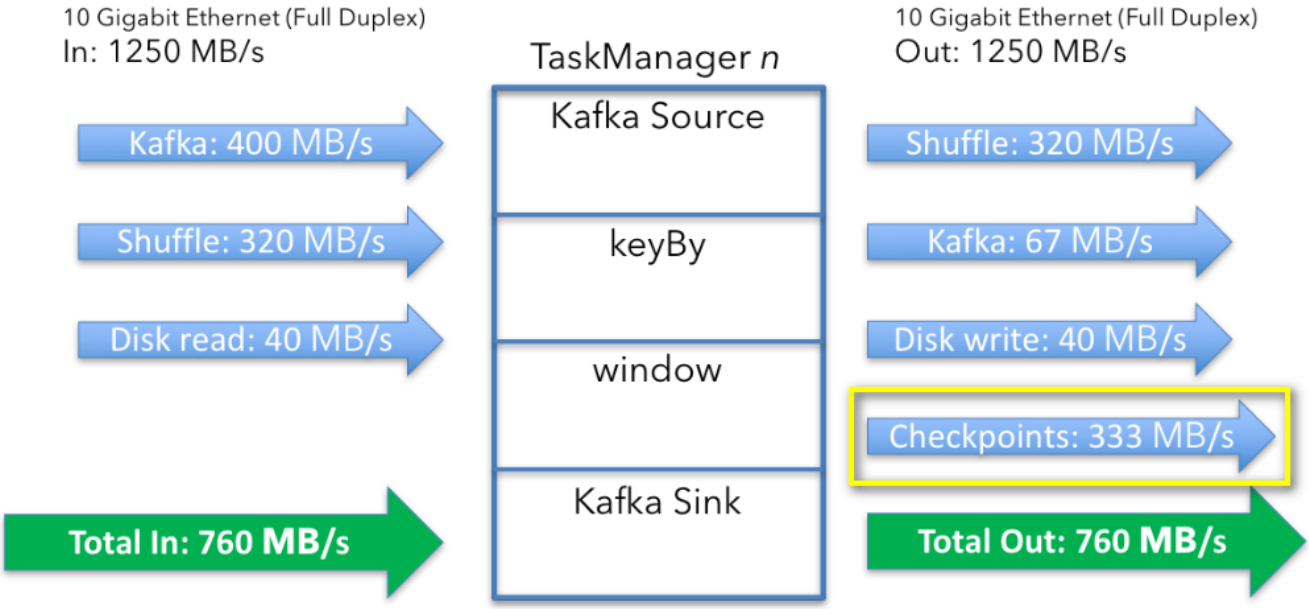
2）checkpoint过程

每当有新event到来上述state过程就会被触发，有时间我们为了保证当任务失败后可以恢复会开启checkpoint，本例中checkpoint设置为每隔一分钟周期性触发，每个checkpoint过程会将现有的state通过网络拷贝到系统中。每个节点一次checkpoint会拷贝的数据为：

40bytes of state x 5 windows x 100,000,000 keys = 20GB

每秒中的数据为20GB ÷ 60 = 333 MB/s。当然checkpoint过程数据同样不是以稳定的速率发送到系统中，而是会以最大的速率发送。此外，从Flink1.3以后，基于RockDB是可以实现增量checkpoint，本例暂时不考虑该特性。单节点整个任务过程网络消耗如下：

- **Data in**: 760MB/s (400 + 320 + 40)
- **Data out**: 760MB/s (320 + 67 + 40 + 333)

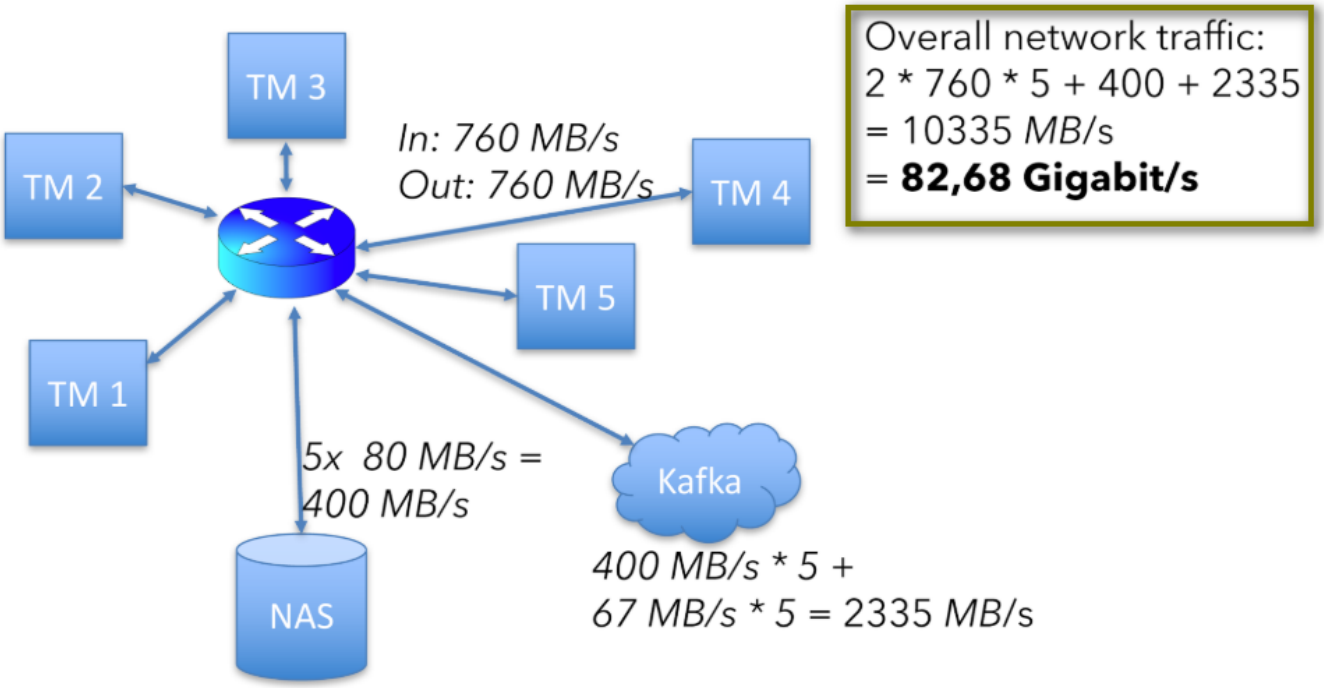


集群整体网络消耗如下：

760 + 760 x 5 + （40×2）×5 + （400+67）×5 = 10335 MB/s

（40×2）×5是5个节点state的读写过程消耗，（400+67）×5是从Kafka读和写过程消耗的（kafka数据会落盘）。

该数据仅为上述硬件设置中的可用网络容量的一半以上，如下图。



2.3.5 总结

该例子中，每个节点流进和流出的数据为760MB/s，仅为节点容量的60%（每个节点为1250MB/s），剩下的40%可以用来应对突发的情况，如网络开销、checkpoint恢复期间的数据重放或者由于数据倾斜导致的节点之间数据shuffle过大的情况等。

回到顶部

3、其他建议

1）CUP个数，Flink官网给出的建议是和slot的个数成比例，从而也就和任务的并行度有关了，换句话说，在考虑任务的并行度时要结合CPU的个数考虑；

2）尽量申请多的内存，内存的最小和可以通过在测试集群中测试后，大致成比例的放大到生成集群中；

3）考虑I/O，数据盘最好和日志盘分离；

4）还有其他如JobManager最好和TaskManager节点分离等；