CANONICAL

在K8s上加速Apache Kafka的7大方法

2021年9月

导言

如果您想要在MicroK8s、Charmed Kubernetes或其他K8s发行版上运行低延迟、高容量的Apache Kafka解决方案，那么您最好花些时间提前规划部署。我们为您提供了一些可能会有所帮助的设计方法，具体取决于您的工作负载。在本白皮书中，我们将介绍七种有助于减少在Kubernetes上运行的高容量、低关键性Kafka解决方案的延迟的技术。

Kubernetes为围绕Apache Kafka等中间件组成的复杂、分布式微服务驱动的应用程序提供了许多好处。此类好处包括改善解决方案的可用性、提高资源利用率、整合服务器以及实现更强健、更可重复和更可预测和部署。

但是，目前还有许多问题需要解决，比如服务基础架构规划、资源调度约束、有状态的服务需求、性能优化以及弹性。同时，根据需求、用例和数据重要性的不同，我们可能需要不同的设计方法。

在本白皮书中，我们将从性能的角度出发，对在Kubernetes服务设计系统上运行Kafka的各个方面进行探索，包括：

1: 您的内存提升

Apache Kafka服务器不会过度占用内存，但Kafka操作系统拥有足够的主内存空间，可用于文件系统页面缓存。理想情况下，所有“热门”主题数据都可存入主存储器。Kafka要求操作系统使用异步fsync调用，间歇性地将日志文件状态刷新到磁盘上。数据被复制并分布于多个Kafka服务器上，并在多台服务器上运行，因此即使是间歇性的fsync调用，单个服务器故障导致数据丢失的风险也相对较低。

主题分区使用者的行为模式可能倾向于随机访问，因此Kafka有足够的RAM，可以将大量的活动工作负载容纳到主存储器中。根据Apache Kafka文档，当前的存储介质对顺序IO进行了高度优化，而随机主存访问操作实际上可能比顺序SSD访问操作还慢。因此，确保工作数据集能够存入主内存对于保证低延迟至关重要。

但是，您可以根据在Kafka中缓冲的数据的重要程度与延迟需求，考虑完全在RAM磁盘（tmpfs 文件系统）上运行Kafka服务器。该操作可在顺序IO操作和随机寻道操作等方面提供卓越的全方位性能，但请注意，使用此架构时，如发生宕机，Kafka主题中所有数据将全部丢失。但是，如前所述，如果是非重要性数据或临时数据，这样的折中操作是可以接受的。

如需将tmpfs卷与Kubernetes的Kafka结合使用，您可使用emptyDir卷类型。请注意，如需为tmpfs卷启用固定大小限制，您应启用相关的特性开关（SizeMemoryBackedVolumes），否则该卷将默认为主机系统内存容量的一半：

```
apiVersion: v1
kind: StatefulSet
metadata:
  name: kafka-broker
spec:
  serviceName: kafka-broker-service
  selector:
    matchLabels:
      app: kafka-broker
  replicas: 7
  template:
    metadata:
      labels:
        app: kafka-broker
    spec:
      containers:
      - image: kafka:2.8
        name: kafka-broker
        volumeMounts:
        - mountPath: /var/kafka/logs
          name: kafka-logs-volume
      volumes:
      - name: kafka-logs-volume
        emptyDir:
          medium: Memory
          sizeLimit: “20Gi”
```

2: 存储选项的考量

如果您不习惯在tmpfs上运行Kafka服务器（可能是因为您的数据具有比此类纯内存方法更高的重要性级别），那么您应该让每个服务器都能访问足够的存储设备（最好是NVME），以便从I/O并行中受益。否则，存储子系统将成为阻碍。Kafka服务器是IO密集型框架，因此集群中的每个节点都应有多个可用的块存储设备。我们建议使用“none”多队列Linux内核IO调度程序配置NVME数据存储设备，并配置XFS文件系统，以实现良好的顺序IO工作负载性能。在合理规模下工作时，XFS还可以提供可靠的随机寻道性能和出色的删除操作性能。

如需将名为“nvme0n1”的块存储设备的内核IO调度程序配置为“none”调度程序，请运行以下命令：

```
echo none | sudo tee /sys/block/nvme0n1/queue/scheduler
```

以下几个RAID配置选项值得商榷：

- 在JBOD（磁盘簇）设置中，Kubernetes Pod可以提供多个块存储设备，并且Kafka服务器下的每个设备都有一个目录
- 在RAID-0配置中，块存储设备被放置在RAID-0（条带化阵列）配置中，并向 Kafka服务器提供目录。
- 在RAID-10（镜像、条带化阵列）配置中，向Kafka服务器提供单个目录。

每种配置都有其利弊。

欢迎您在微博上的Ubuntu、MicroK8s和Kafka项目示例中提及我们（@ubuntu）——我们希望能够了解这些项目！

JBOD的配置过程非常简单。但是，JBOD可以与Kafka服务器的内部主题分区平衡机制进行次优配合。RAID-0的性能比其他RAID级别更加出色，但可能依旧不如JBOD，具体取决于RAID控制器；此外，如果阵列中单个存储设备出现故障，阵列中保存的所有数据都将丢失。但是，这种情况或许是可以接受的，具体取决于管理主题的复制因子和同步副本配置，或者数据的重要性。RAID-10提供最佳的数据丢失保护；但这种保护是以存储效率和性能为代价，因为数据将进行跨存储阵列备份以及在Kafka服务器之间复制。

对于Kubernetes来说，公开本地存储卷可能是一种有效的方法。由于本地存储卷具有持久性并且与单个主机物理关联，因此需要nodeAffinity（节点亲和性）规则来确保pod调度的一致性。这样做的好处是，除了计划用于Kafka的节点以外，其他K8s集群中的节点均不需要在阵列中配置大量的存储设备。之后，Kubernetes调度程序会将Kafka服务器分配给满足Kafka服务器pod调度标准的工作节点。

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv-microk8s-0-1
spec:
  capacity:
    storage: 250Gi
  volumeMode: Filesystem
  accessModes:
  - ReadWriteOnce
  persistentVolumeReclaimPolicy: Delete
  storageClassName: local-storage
  local:
    path: /mnt/disks/ssd1
  nodeAffinity:
    required:
      nodeSelectorTerms:
      - matchExpressions:
        - key: kubernetes.io/hostname
          operator: In
          values:
          - microk8s-0
```

3: 您需要复制多少?

Kafka使用复制来确保主题中数据的高可用性。可以使用以下两个关键选择：主题复制因子和最小同步副本计数。复制因子指在Kafka集群中每条消息应保存的副本数量，而最小同步副本是指在既定时刻需要保证的副本数量与主题分区Leader保持同步。请注意，每个主题都可以配置此类参数。这些数值可在延迟/性能和耐久性之间进行权衡。

对于本质上是完全短暂的高容量、低延迟工作负载（即需要近乎实时处理但未计划进行长期处理的流式数据），最好最大程度地减少复制开销，并仅为每个主题分区配置一个副本。但是，对于关注数据丢失且更为关键的工作负载，复制消息绝对有意义。但请记住，跨集群中复制数据可能会显著增加开销，对于某些高容量的工作负载，可能需要进行取舍。

如需调整主题分区"foo: 0"的复制因子，以便仅有一个副本托管在服务器0上，请运行以下指令：

```
cat > adapt-replication-factor.json <<EOF
{
  “version”:1,
  “partitions”:[
  {
   “topic”:”foo”,
   “partition”:0,
   “replicas”:[0]
  }]
}
EOF

$KAFKA_HOME/bin/kafka-reassign-partitions.sh \
--bootstrap-server $KAFKA_BROKER_HOSTNAME:9092 \
--reassignment-json-file \
adapt-replication-factor.json --execute
```

如需为主题"foo"调整最小同步副本，以便每个主题分区仅有一个副本，请运行以下指令：

```
$KAFKA_HOME/bin/kafka-topics.sh \
--alter --bootstrap-server \
$KAFKA_BROKER_HOSTNAME:9092 --topic foo \
--config min.insync.replicas=0
```

4: 保留策略的加强

虽然从理论上讲，长期保存主题数据不会影响Kafka集群的整体性能，但在某些情况下，它可能会对集群性能造成不利影响。在高容量、低延迟的环境中，应将工作数据集保留在集群中每个服务器的内存中（通常通过操作系统文件系统页面缓存）。如果工作数据集大小超过操作系统的缓存能力，则性能可能会显著下降。

试想一下，在32GB RAM的系统上，每个服务器每小时保存在Kafka中的消息量共约为100GB。在此情况下，操作系统的页面缓存将完全饱和，整体性能可能会因此受到影响。同样地，如果部署配置为使用15GB tmpfs卷作为备份，则该解决方案将在大约9分钟后超出存储容量。

因此，为主题配置的保存策略是另一种调整方式，这对解决方案的整体性能和稳定性会产生重大影响。如果您的目标是在没有长期持久性需求的高容量、非关键的工作负载上进行低延迟处理，那么将主题保存策略调整到较短时长和相对较小的最大大小可能很有帮助。在这种情况下，需要在消费者能够及时处理消息和系统无法跟上消息生成速度之间进行权衡。我们在这里采取的设计选择是删除无法被快速处理的消息，以便随时了解生成者的最新消息。

保存策略可以定义于Kafka服务器级在传递给Kafka服务器过程的一组Java属性或属性文件中，也可定义于使用CLI的主题中。如需更改保留策略，以存储最长为5分钟或最大为15GB的消息（以最快达到的为准），对于主题“foo”，请运行以下指令：

```
$KAFKA_HOME/bin/kafka-configs.sh \
--bootstrap-server $KAFKA_BROKER_HOSTNAME:9092 \
--alter --entity-type topics --entity-name foo \
--add-config ‘retention.ms=300000’

$KAFKA_HOME/bin/kafka-configs.sh \
--bootstrap-server $KAFKA_BROKER_HOSTNAME:9092 \
--alter --entity-type topics --entity-name foo \
--add-config ‘retention.bytes=16106119114’
```

5: 降低消息大小

人们普遍认为，Kafka的性能得益于较小的消息大小，单个消息最佳的大小远低于1MB。因此，在高容量、低延迟的情况下，优化工作负载使单个消息尽可能变小，从而可以显著提高整体性能。

试想一下，如果Kafka解决方案每秒大约接收500条消息，每条消息的大小是150KB。这意味着该解决方案每小时需要处理约270GB的数据。通过压缩、更改编码选择和/或更高效的数据结构将消息大小降至75KB，即该解决方案需要处理一半的数据量——每小时135GB的数据，这可以大幅减小解决方案的规模，或可能有助于更严格地遵循具有相同基准的性能要求。

Apache Kafka消息中有一种流行且高效的数据编码格式——Apache Avro。通常与Kafka集成的另一种数据编码格式是协议缓冲区，其可能具有更高的性能。

6: 考虑简化配置

Kafka带有许多旋钮和拨盘，可在传输过程中支持高数据质量和高耐久性。例如，“精确执行一次”消息传递语义、生成者“确认”、发生瞬时错误的生成者重试。由于此类功能可能会对整体系统性能产生显著影响，因此有必要评估其对您的场景的必需性。如果数据的关键性较低、消息量较大，并且总体延迟非常重要，那么以部分重复、耐久性降低或消息丢失换取更好的整体系统性能可能很有意义。

下列的Java码本展示了禁用确认和重试的示例生成者配置：

```
public class SomeKafkaProducer {

    private static Producer<String, Something>
        createProducer() {

        final Properties props = new Properties();
        setupProperties(props);
        // ...
        return new KafkaProducer<>(props);
    }

    private static void setupProperties(
        Properties props) {

        props.put(ProducerConfig.RETRIES_CONFIG, 0);
        props.put(ProducerConfig.ACKS_CONFIG,
            “none”);
    }
}
```

7: 分区的魔法

Kafka主题分区通常是从主题消费者的视角来看的，但是，主题分区也可以通过类似地将工作负载分布到多个服务器，从而使生成者受益。每个主题分区都分配有一个Kafka服务器作为Leader，因此在数据关键性低但延迟要求高的高容量情况下，仅为每个主题分区配置一个副本意味着生成者可以在集群中的服务器之间均匀地分配主题写入，且完全没有任何复制开销。同样地，在配置复制时，当集群中有多个节点，主题分区仍然有利。例如，在由9个Kafka服务器组成的集群中，如果每个主题分区以三种方式进行复制，则可以在九个可能的节点中的三个节点之间分配每个主题分区副本。因此，生成者工作负载并行拥有足够的空间。

如需将名为“foo”主题的主题分区数设置为20个，请运行以下指令：

```
$KAFKA_HOME/bin/kafka-topics.sh \
--bootstrap-server $KAFKA_BROKER_HOSTNAME:9092 \
--alter --topic foo --partitions 20
```

总结

在本白皮书中，我们提出了在Kubernetes上规划部署高容量、低延迟的Apache Kafka时可以考虑的七种方法：

1 确保您有足够的内存。
2 您的持续存储选项的考量。
3 仔细评估 Topic 复制的需求。
4 工作负载的保留策略的调整。
5 让消息大小降到最低。
6 尽可能让您的broker配置简化。
7 利用分区的优势。

您是否知道Canonical可以提供完全托管的服务来帮助您启动和运行Apache Kafka，以及部分其他流行的开源应用程序？我们可以根据您的需求设计、构建和操作Kafka，并在您准备接管时进行过渡操作。马上联系我们，我们会安排会议讨论您的需求

资源列表

- 了解更多关于Apache KafkaCanonical Managed Apps服务的信息
- 全方位了解Canonical Managed Apps服务
- 联系销售代表

ubuntu
Delivered by Canonical