消息队列 CKafka

扩展阅读

产品文档

腾讯云

文档目录

扩展阅读
CKafka 数据可靠性说明

最近更新时间：2019-07-12 20:40:41

现有的技术无法保证数据100%不丢失，您可以通过一些配置参数最大可能保证数据高可靠。
本文将分别通过生产端、服务端（CKafka）和消费端介绍影响消息队列 CKafka 可靠性的因素，并提供对应的解决方法。

生产端

数据丢失原因

生产者将数据发送到消息队列 CKafka 时，数据可能因为网络抖动而丢失，此时消息队列 CKafka 未收到该数据。可能情况：

- 网络负载高或者磁盘繁忙时，生产者又没有重试机制。
- 磁盘超过购买规格的限制，例如实例磁盘规格为9000GB，在磁盘写满后未及时扩容，会导致数据无法写入到消息队列 CKafka。
- 突发或持续增长峰值流量超过购买规格的限制，例如实例峰值吞吐规格为100MB/s，在长时间峰值吞吐超过限制后未及时扩容，会导致数据写入消息队列 CKafka 变慢，生产者有排队超时机制时，导致数据无法写入到消息队列 CKafka。

解决方法

- 生产者对自己重要的数据，开启失败重试机制。
- 针对磁盘使用，在配置实例时设置好监控和 告警策略 ，可以做到事先预防。
  遇到磁盘写满时，可以在控制台及时升配（消息队列 CKafka 非独占实例间升配为平滑升配不停机且也可以单独升配磁盘）或者通过修改消息保留时间降低磁盘存储。
- 为了尽可能减少生产端消息丢失，您可以通过 buffer.memory 和 batch.size（以字节为单位）调优缓冲区的大小。缓冲区并非越大越好，如果由于某种原因生产者 done 掉了，那么缓冲区存在的数据越多，需要回收的垃圾越多，恢复就会越慢。应该**时刻注意生产者的生产消息数情况、平均消息大小**等（消息队列 CKafka 监控中有丰富的监控指标）。
- 配置生产端 ACK
  当 producer 向 leader 发送数据时，可以通过 request.required.acks 参数以及min.isync.replicas 设置数据可靠性的级别。
  - 当 acks = 1时（默认值），生产者在 ISR 中的 leader 已成功收到数据可以继续发送下一条数据。如果 leader 宕机，由于数据可能还未来得及同步给其 follower，则会丢失数据。
  - 当 acks = 0时，生产者不等待来自 broker 的确认就发送下一条消息。这种情况下数据传输效率最高，但数据可靠性确最低。

- 当 acks = -1或者 all 时，生产者需要等待 ISR 中的所有 follower 都确认接收到消息后才能发送下一条消息，可靠性最高。
  即使按照上述配置 ACK，也不能保证数据不丢，例如，当 ISR 中只有 leader 时（ISR 中的成员由于某些情况会增加也会减少，最少时只剩一个 leader），此时会变成 acks = 1的情况。所以需要同时在配合 min.isync.replicas 参数（此参数可以在消息队列 CKafka 控制台 Topic 配置开启高级配置中进行配置），min.insync.replicas 表示在 ISR 中最小副本的个数，默认值是1，当且仅当 acks = -1或者 all 时生效。

建议配置的参数值

此参数值仅供参考，实际数值需要依业务实际情况而定。

- 重试机制： `message.send.max.retries=3;retry.backoff.ms=10000;`
- 高可靠的保证： `request.required.acks=-1;min.isync.replicas=2;`
- 高性能的保证： `request.required.acks=0;`
- 可靠性+性能： `request.required.acks=1;`

服务端（CKafka）

数据丢失原因

- partition 的 leader 在未完成副本数 followers 的备份时就宕机，即使选举出了新的 leader 但是数据因为未来得及备份就丢失。
- 开源 Kafka 的落盘机制为异步落盘，也就是数据是先存在 PageCache 中的，当还没有正式落盘时，broker 出现断点或者重启或者故障时，PageCache 上的数据由于没有来及落磁盘进而丢失。
- 磁盘故障导致已经落盘的数据丢失。

解决方法

- 开源 Kafka 是多副本的，官方推荐通过副本来保证数据的完整性，此时如果是多副本，同时出现多副本多 broker 同时挂掉才会丢数据，比单副本数据的可靠性高很多，所以消息队列 CKafka 强制 Topic 是双副本，可配置3副本。
- 消息队列 CKafka 服务配置了更合理的参数 log.flush.interval.messages 和 log.flush.interval.ms，对数据进行刷盘。
- 消息队列 CKafka 对磁盘做了特殊处理，保证部分磁盘损坏时也不会影响数据的可靠性。

建议配置的参数值

非同步状态的副本可以选举为 leader： `unclean.leader.election.enable=false // 关闭`

消费端

数据丢失原因

- 还未真正消费到数据就提交 commit 了 offset，若过程中消费者挂掉，但 offset 已经刷新，消费者错过了一条数据，需要消费分组重新设置 offset 才能找回数据。

- 消费速度和生产速度相差太久，而消息保存时间太短，导致消息还未及时消费就被过期删除。

解决方法

- 合理配置参数 auto.commit.enable，等于 true 时表示自动提交。建议使用定时提交，避免频繁 commit offset。
- 监控消费者的情况，正确调整数据的保留时间。监控当前消费 offset 以及未消费的消息条数，并配置告警，防止由于消费速度过慢导致消息过期删除。

CKafka 压测指南

最近更新时间：2019-04-15 15:37:47

测试工具

Kafka Producer 和 Consumer 的性能测试均可使用 Kafka 客户端自带的开源脚本，主要输出每秒发送消息量（MB/second）和每秒发送消息数（records/second）两项指标。

- Kafka Producer 测试脚本：`$KAFKA_HOME/bin/kafka-producer-perf-test.sh`
- Kafka Consumer 测试脚本：`$KAFKA_HOME/bin/kafka-consumer-perf-test.sh`

测试命令

生产测试命令示例：

```
bin/kafka-producer-perf-test.sh
--topic test
--num-records 123
--record-size 1000
--producer-props bootstrap.servers= ckafka vip : port
--throughput 20000
```

消费测试命令示例：

```
bin/kafka-consumer-perf-test.sh
--topic test
--new-consumer
--fetch-size 10000
--messages 1000
--broker-list bootstrap.servers=ckafka vip : port
```

测试建议

- 为了提高吞吐量，建议创建分区时数量 ≥ 3（因后端 CKafka 集群节点数量最少是3，如只创建1个分区则分区会分布在一个 Broker 上面，影响性能）。
- 由于 CKafka 是分区级别消息有序的，因此过多的分区也会影响生产性能，根据实际压测，建议分区数不超过6。
- 为了保证压力测试的效果，需要多客户端模拟一定的并发，建议采用多台机器作为压测客户端（生产端），每台启动多个压测程序，提高并发。此外建议每1s启动一个生产者，避免同时启动所有生产者导致测试机器高负载。

CKafka 常用参数配置指南

最近更新时间：2019-04-15 15:39:29

Broker 配置参数说明

当前 CKafka broker 端的一些配置如下，供参考：

```
# 消息体的最大大小，单位是字节
message.max.bytes=1000012

# 是否允许自动创建 topic，默认是 false，当前可以通过控制台或云 API 创建。
auto.create.topics.enable=false

# 是否允许调用接口删除 Topic
delete.topic.enable=true

# Broker 允许的最大请求大小为 16MB
socket.request.max.bytes=16777216

# 每个 IP 与 Broker 最多建立 5000 个连接
max.connections.per.ip=5000

# offset 保留时间，默认为 7 天
offsets.retention.minutes=10080

# 没有 ACL 设置时，允许任何人访问
allow.everyone.if.no.acl.found=true

# 日志分片大小为 1G
log.segment.bytes=1073741824

# 日志滚动检查间隔 5 分钟，当设置保留时间小于 5 分钟时，也可能需要等待 5 分钟才会清空日志。
log.retention.check.interval.ms=300000
```

注意：其他未列出的 Broker 配置参考 开源 Kafka 默认配置。

Topic 配置参数说明

1. 选取合适的分区数量

从生产者的角度来看，向不同的 partition 写入是完全并行的；从消费者的角度来看，并发数完全取决于 partition 的数量（如果 consumer 数量大于 partition 数量，则必有 consumer 闲置）。因此选取合适的分区数量对于发挥 CKafka 实例的性能十分重要。
partition 的数量需要根据生产和消费的吞吐来判断。理想情况下，可以通过如下公式来判断分区的数目：

```
Num = max( T/PT , T/CT ) = T / min( PT , CT )
```

其中，Num 代表 partition 数量，T 代表目标吞吐量，PT 代表生产者写入单个 partition 的最大吞吐，CT 代表消费者从单个 partition 消费的最大吞吐。则 partition 数量应该等于 T/PT 和 T/CT 中较大的那一个。

在实际情况中，生产者写入但 partition 的最大吞吐 PT 的影响因素和批处理的规模、压缩算法、确认机制、副本数等有关。消费者从单个 partition 消费的最大吞吐 CT 的影响因素和业务逻辑有关，需要在不同场景下实测得出。

通常建议 partition 的数量一定要大于等于消费者的数量来实现最大并发。 如果消费者数量是 5，则 partition 的数目也应该是 >=5 的。同时，过多的分区会导致生产吞吐的降低和选举耗时的增加，因此也不建议过多分区。提供如下信息供参考：

- 单个 partition 是可以实现消息的顺序写入的。
- 单个 partition 只能被同消费者组的单个消费者进程消费。
- 单个消费者进程可同时消费多个 partition，即 partition 限制了消费端的并发能力。
- partition 越多则失败后 leader 选举的耗时越长。
- offset 的粒度最细是在 partition 级别的，partition 越多，查询 offset 就越耗时。
- partition 的数量是可以动态增加的，只能增加不能减少。但增加会出现消息 rebalance 的情况。

2. 选取合适的副本

目前为了保证可用性副本数必须大于等于 2，如果需要保障高可靠建议 3 副本。

> **注意：**副本数会影响生产/消费流量，如 3 副本则实际流量= 生产流量*3

3. 日志保留时间

Topic 的 log.retention.ms 配置通过控制台实例的保留时间统一设置。

4. 其他 Topic 级别配置说明

```
# Topic 级别最大消息大小
max.message.bytes=1000012

# 0.10.2 版本消息格式为 V1 格式
message.format.version=0.10.2-IV0
```

```
# 不在 ISR 中的 replica 允许选择为 Leader，可用性高于可靠性，存在数据丢失风险。
unclean.leader.election.enable=true

# ISR 提交生产者请求的最小副本数。如果同步状态的副本数小于该值，服务器将不再接受。request.required.acks为-1或all的写入请求。
min.insync.replicas=1
```

生产者配置指南

生产端常用参数配置如下，建议客户根据实际业务场景调整配置：

```
# 生产者会尝试将业务发送到相同的 Partition 的消息合包发送到 Broker，batch.size 设置合包的大小上限。默认为 16K。batch.size 设太小会导致吞吐下降，设太大会导致内存使用过多。
batch.size=16384

# Kafka producer 的 ack 有 3 种机制，分别说明如下：
# -1 或 all：Broker 在 leader 收到数据并同步给所有 ISR 中的 follower 后，才应答给 Producer 继续发送下一条（批）消息。 这种配置提供了最高的数据可靠性，只要有一个已同步的副本存活就不会有消息丢失。注意：这种配置不能确保所有的副本读写入该数据才返回，可以配合 Topic 级别参数 min.insync.replicas 使用。
# 0：生产者不等待来自 broker 同步完成的确认，继续发送下一条（批）消息。这种配置生产性能最高，但数据可靠性最低（当服务器故障时可能会有数据丢失，如果 leader 已死但是 producer 不知情，则 broker 收不到消息）
# 1：生产者在 leader 已成功收到的数据并得到确认后再发送下一条（批）消息。这种配置是在生产吞吐和数据可靠性之间的权衡（如果leader已死但是尚未复制，则消息可能丢失）

# 用户不显示配置时，默认值为 1。用户根据自己的业务情况进行设置。
acks=1

# 控制生产请求在 Broker 等待副本同步满足 acks 设置的条件所等待的最大时间。
timeout.ms=30000

# 配置生产者用来缓存消息等待发送到 Broker 的内存。用户要根据生产者所在进程的内存总大小调节。
buffer.memory=33554432

# 当生产消息的速度比 Sender 线程发送到 Broker 速度快，导致 buffer.memory 配置的内存用完时会阻塞生产者 send 操作，该参数设置最大的阻塞时间。
max.block.ms=60000

# 客户端在每个连接上最多可发送的最大的未确认请求数，该参数大于 1 且 retries 大于 0 时可能导致数据乱序。
max.in.flight.requests.per.connection=5

# 设置消息延迟发送的时间，这样可以等待更多的消息组成 batch 发送。默认为 0 表示立即发送。当待发送的消息达到 batch.size 设置的大小时，不管是否达到 linger.ms 设置的时间，请求也会立即发送。
linger.ms=0

# 生产者能够发送的请求包大小上限，默认为 1M。在修改该值时注意不能超过 Broker 配置的包大小上限 16M。
```

```
max.request.size=1048576

# 压缩格式配置，目前 0.9(包含)以下版本不允许使用压缩，0.10（包含）以上不允许使用 GZip 压缩。
compression.type=[none, snappy, lz4]

# 客户端发送给 Broker 的请求的超时时间，不能小于 Broker 配置的 replica.lag.time.max.ms，目前该值为 10000 ms。
request.timeout.ms=30000

# 请求发生错误时重试次数，当 retries 大于 0 且 max.in.flight.requests.per.connection 大于 1 时可能会导致乱序。建议客户将该值改成大于 0，即重复发送失败的消息。
retries=0

# 发送请求失败时到下一次重试请求之间的时间。
retry.backoff.ms=100
```

消费者配置指南

生产端常用参数配置如下，建议客户根据实际业务场景调整配置：

```
# 是否在消费消息后将 offset 同步到 Broker，当 Consumer 失败后就能从 Broker 获取最新的 offset。
auto.commit.enable=true

# 当 auto.commit.enable=true 时，自动提交 Offset 的时间间隔，建议设置至少 1000。
auto.commit.interval.ms=5000

# 当 Broker 端没有 offset（如第一次消费或 offset 超过 7 天过期）时如何初始化 offset，当收到 OFFSET_OUT_OF_RANGE 错误时，如何重置 Offset。
# earliest：表示自动重置到 partition 的最小 offset。
# latest：默认为 latest，表示自动重置到 partition 的最大 offset。
# none：不自动进行 offset 重置，抛出 OffsetOutOfRangeException 异常。
auto.offset.reset=latest

# 标识消费者所属的消费分组
group.id=""

# 使用 Kafka 消费分组机制时，消费者超时时间。当 Broker 在该时间内没有收到消费者的心跳时，认为该消费者故障失败，Broker 发起重新 Rebalance 过程。目前该值的配置必须在 Broker 配置。group.min.session.timeout.ms=6000和group.max.session.timeout.ms=300000 之间
session.timeout.ms=10000

# 使用 Kafka 消费分组机制时，消费者发送心跳的间隔。这个值必须小于 session.timeout.ms，一般小于它的三分之一
heartbeat.interval.ms=3000
```

```
# 使用 Kafka 消费分组机制时，再次调用 poll 允许的最大间隔。如果在该时间内没有再次调用 poll，则认为该消费者已经失败，Broker 会重新发起 Rebalance 把分配给它的 partition 分配给其他消费者
max.poll.interval.ms=300000

# Fecth 请求最少返回的数据大小。默认设置为 1B，表示请求能够尽快返回。增大该值会增加吞吐，同时也会增加延迟
fetch.min.bytes=1

# Fetch 请求最多返回的数据大小，默认设置为 50MB
fetch.max.bytes=52428800

# Fetch 请求等待时间
fetch.max.wait.ms=500

# Fetch 请求每个 partition 返回的最大数据大小，默认为 10MB
max.partition.fetch.bytes=1048576

# 在一次 poll 调用中返回的记录数
max.poll.records=500

# 客户端请求超时时间，如果超过这个时间没有收到应答，则请求超时失败
request.timeout.ms=305000
```

CKafka 客户端常见异常说明

最近更新时间：2019-04-15 15:38:54

Kafka Java Client Exception

以下异常属于客户端配置或服务异常，客户端不会自动重试。

异常	描述	分析与说明
UnknownServerException	服务器处理请求发生未知错误。	老版本流控会返回这个错误；新版本则可能是服务器出现 BUG 导致。
RecordTooLargeException	消息太大。	目前配置 message.max.bytes=1000012。
InvalidRequiredAcksException	生产者配置的 acks 参数不合法。	
InconsistentGroupProtocolException	Group 的协议不一致。	检查 Consumer 和 Connector 是否配置了相同的 group.id，这两者使用的不同的协议，不能加入相同的组。
InvalidGroupIdException	Consumer Group ID 不合法。	建议使用 [a-zA-Z0-9._-] 这些字符，长度不超过 128。
InvalidTopicException	Topic 不合法。	开启自动创建 Topic 选项后，客户端使用的 Topic 不合法会返回这个异常。检查 Topic 是否使用了不合法的字符或长度是否超过限制。
InvalidSessionTimeoutException	消费者配置的 session.timeout.ms 不合法。	目前服务器端允许的最小值为：group.min.session.timeout.ms=6000，最大值为：group.max.session.timeout.ms=300000。
InvalidCommitOffsetSizeException	提交 Offset 信息太大超过最大消息大小，无法写入 _*consumer*offsets。	目前配置 message.max.bytes=1000012。
OffsetMetadataTooLarge	Offset 提交请求包含的 Metadata 太大。	服务器配置的 offset.metadata.max.bytes=4096。
UnsupportedVersionException	Broker 不支持该版本的请求。	建议使用 0.10.2.x 版本的客户端。

以下异常在程序正常运行过程中可能会短暂出现，客户端会自动重试。 持续出现则服务不正常。

异常	描述	分析与说明
TimeoutException	请求超时。	首次连接报请求超时，先检查地址是否正确，telnet 确定网络能够联通。程序运行中偶尔抛出此异常可能属于网络抖动。
CorruptRecordException	消息不合法。	可能 CRC 检查不通过，数据大小不合法。此外，如果压缩方式使用 **GZip** 或 0.9 以下版本 **使用压缩** 也会导致这个错误。
UnknownTopicOrPartitionException	Topic 或 Partition 不存在。	到控制台检查是否已经创建对应的 Topic。注意：客户端通过 TopicName生产消费，而不是 TopicId。此外，客户端没有权限访问 Topic 时也会报 Topic 不存在。
LeaderNotAvailableException	Partition 没有 Leader。	当 Topic 刚创建时服务器还未选出合适的 Leader，此时会返回此错误给客户端，客户端会自动重试获取 Leader 信息。**旧版本才会有这个异常，0.10.2.1 已经去掉。**
NotLeaderForPartitionException	Partition 的 Leader 不可用。	由于客户端会缓存 Topic 的 Metadata，所以当 Partition 的 Leader 切换时，生产或消费请求可能仍然发送到旧 Leader 上，此时会返回此错误给客户端，客户端会自动更新 Metadata 信息。
NetworkException	客户端连接被服务器端关闭。	网络异常或连接数超过限制。
NotEnoughReplicasException	ISR 数量不够。	在写入数据时 Partition 的 ISR 数量小于 Topic 配置的 min.insync.replicas，可能由于 ISR 抖动导致。
NotEnoughReplicasAfterAppendException	数据写入 Broker 本地后，发生 ISR 抖动导致无法满足 min.insync.replicas。	

以下异常在日志配置为 DEBUG 级别会出现，客户端会自动处理。

异常	描述	分析与说明

异常	描述	分析与说明
OffsetOutOfRangeException	消费者拉取消息时传入的 Offset 超出范围。	如果客户端设置了 Offset 重置策略（earliest 或 latest），则客户端会根据策略进行 Offset 重置，否则需要用户程序处理这个异常
GroupLoadInProgressException	ConsumerGroup 对应的 Coordinator正在加载。	服务器端升级时可能短暂出现，客户端会自动重试。
GroupCoordinatorNotAvailableException	Coordinator 不可用。	服务器端升级时可能短暂出现，客户端会自动重试。
NotCoordinatorForGroupException	当前节点不是该ConsumerGroup的Coordinator，Coordinator 迁移到别的节点。	服务器端升级时可能短暂出现，客户端会自动重试。
IllegalGenerationException	ConsumerGroup 的generation不合法。	可能心跳超时或有新消费者加入，Consumer 会自动重新尝试加入ConsumerGroup。
RebalanceInProgressException	ConsumerGroup 正在进行 rebalance。	可能心跳超时或有新消费者加入，Consumer 会自动重新尝试加入ConsumerGroup。