KIP-427: Add AtMinIsr topic partition category (new metric & TopicCommand option)

Status

Current state: *ACCEPTED*

Discussion thread: here

Vote thread: here

JIRA: *KAFKA-7904*

PR*: https://github.com/apache/kafka/pull/6421*

Motivation

Today, a topic partition may be categorized as:

(1) Fully in sync (inSyncReplicas == allReplicasMap)

(2) UnderReplicated

(3) UnderMinIsr

(4) Offline

(3) and (4) are failure scenarios in which clients will face unavailability (producers with acks=ALL will fail to produce if ISR count is under the configured "min.insync.replicas" count).

(2) Under-replicated partitions occur whenever the inSyncReplicas set is not equal to the allReplicasMap, which can happen when we have:

- Repartitioning
- Broker restarts
- Transient network issues
- Broker failures

The current categorization of topic partitions has a gap as an UnderReplicatedPartition is generic and triggered in the various situations listed above. This makes it incredibly difficult to use UnderReplicatedPartitions as an indicator for alerting as alerts configured on this metric will trigger whenever there is a change in the ISR set.

In reality, we can actually tolerate a reduced ISR set **as long as it meets the minimum insync replica count** (as configured by the "min.insync.replicas" configuration), otherwise producers with acks=ALL configured will fail.

This KIP aims to improve monitoring by proposing a new metric group **AtMinIsr,** which consists of partitions that only have the minimum number of insync replicas remaining in the ISR set (as configured by "min.insync.replicas"). This new metric can be used as a warning since any partitions that are AtMinIsr are at danger of causing at least producer unavailability (for acks=ALL producers) if one more replica drops out of the ISR set.

We can first objectively define UnderReplicated and AtMinIsr that applies in all scenarios:

UnderReplicated: A partition in which the isr_set_size is not equal to the replica_set_size (isr_set_size can be bigger or smaller than replica_replication_factor)
AtMinIsr: A partition in which the isr_set_size is equal to the min_isr_size, which also means 1 more drop in isr_set_size will lead to at least producer (acks=ALL) failure

Examples

Let's first take a look at how UnderReplicated and AtMinIsr change depending on different configurations. All examples below look at 1 partition.

Example 1 (min_isr_size = replica_replication_factor - 1):

replica_replication_factor = 3

min_isr_size = 2

ISR set = [0,1,2]

1. Broker 0 fails

- ISR set = [1,2]
- partition is UnderReplicated and AtMinIsr

2. Broker 1 fails

- ISR set = [2]
- partition is UnderMinIsr

In this example, AtMinIsr has the same outcome as UnderReplicated.

Example 2 (min.insync.replicas = 1 AND replica_replication_factor > 2):

replica_replication_factor = 3

min_isr_size = 1

ISR set = [0,1,2]

1. Broker 0 fails

- ISR set = [1,2]
- partition is UnderReplicated

2. Broker 1 fails

- ISR set = [2]
- partition is UnderReplicated and AtMinIsr

3. Broker 2 fails

- ISR set = []
- partition is OfflinePartition

In this example, AtMinIsr triggers when there is only 1 insync replica remaining, and tells us that 1 more failure will cause producers with ack=ALL to be unavailable (the whole partition will be unavailable in this scenario).

Example 3 (replication-factor - min.insync.replicas > 1):

replica_replication_factor = 4

min_isr_size = 2

ISR set = [0,1,2,3]

1. Broker 0 fails

- ISR set = [1,2,3]
- partition is UnderReplicated

2. Broker 1 fails

- ISR set = [2,3]
- partition is UnderReplicated and AtMinIsr

3. Broker 2 fails

- ISR set = [3]
- partition is UnderMinIsr

In this example, AtMinIsr triggers when there is the min_isr_size remaining in the ISR set, while UnderReplicated triggers after the first failure.

Advantages Over UnderReplicatedPartition Metric

In some scenarios such as Example 1 where min_isr_size = replica_replication_factor - 1, the AtMinIsr metric is the exact same as the UnderReplicated metric.

However, here are a few scenarios in which AtMinIsr provides an improvement over UnderReplicated:

(1) Repartitioning

When an admin triggers a repartition, the ISR set is first expanded from [old_set] to [old_set + new_set], and then reduced to just the [new_set]. In this case, UnderReplicated will be non-zero even when the ISR set is [old_set + new_set]. AtMinIsr will not be non-zero during [old_set + new_set] step unless something goes wrong during repartitioning and replicas are failing to fetch (reducing the isr_set_size to min_isr_size), but we want to know if this happens.

(2) min.insync.replicas = 1 and replication-factor > 2

The default value for this configuration is 1, and users can change this to provide higher durability guarantees. In the default scenario where min.insync.replicas = 1 and replication-factor = 3, the AtMinIsr metric will be non-zero when isr_set_size = 1, which tells us that 1 more drop in this set will lead to a completely unavailable partition. This is very powerful for users that have min.insync.replicas = 1 and replication-factor > 2.

(3) replication-factor - min.insync.replicas > 1

Kafka is built to be fault-tolerant, so we ideally want to be able to tolerate more than 1 failure which means we want the difference between replication-factor and min.insync.replicas to be > 1. If it is equal to 1, then we can only tolerate 1 failure otherwise acks=ALL producers will fail.

We generally want isr_set_size to equal replica_replication_factor to have the best guarantees, but this is not always possible for all Kafka users depending on their environment and resources. In some situations, we can allow the isr_set_size to be reduced, especially if we can tolerate more than 1 failure (replication-factor - min.insync.replicas > 1). The only requirement is that the isr_set_size must be at least min_isr_size otherwise acks=ALL producers will fail.

One example is if we have a cluster with massive load and we do not want to trigger a repartition to make isr_set_size = replica_replication_factor unless absolutely necessary as repartitioning introduces additional load which can impact clients. Maybe we also expect the failed broker to be restored soon so we don't want to do anything unless absolutely necessary. In these scenarios, the AtMinIsr metric will tell us when we absolutely need to consider repartitioning or some other action to restore the health of the cluster (false negative is still possible but it tells us that we could not tolerate any more failure at the time it was non-zero if we do not want acks=ALL producers to fail).

Usage

A potential usage of this new AtMinIsr category is:

1. Set up an alert to trigger when AtMinIsr > 0 for a period of time
2. If the alert is triggered, then assess the health of the cluster:
    a. If there is an ongoing maintenance, then no action is needed
    b. Otherwise a broker may be unhealthy. The AtMinIsr partition metric or --at-min-isr-partitions TopicCommand option can be used to determine the list of topics to repartition if the unhealthy broker(s) cannot be fixed quickly

NOTE: Alerts on OfflinePartition and UnderMinIsr should still be configured as AtMinIsr may not be triggered if the ISR set state goes directly to OfflinePartition/UnderMinIsr.

AtMinIsr Values + Possible Explanations

1. AtMinIsr is zero

Everything is fine, and business as usual. Nothing to do here.

2. AtMinIsr is consistently greater than zero for a prolonged period of time

Broker(s) may have failed so this could warrant an alert for an operator to take a look at the health of the cluster to see if any brokers are downed.

3. AtMinIsr bounces between zero and non-zero repeatedly

Broker(s) may be experiencing trouble such as high load or temporary network issues which is causing the partition to temporarily fall out of sync.

NOTE: There are still scenarios in which AtMinIsr will be non-zero during planned maintenance. For example, if RF=3 and minIsr is set to 2, then a planned of a broker can cause AtMinIsr to be non-zero. This however should not be occurring outside of planned maintenance.

Public Interfaces

We will introduce two new metrics and a new TopicCommand option to identify AtMinIsr partitions.

```
# New metrics
# Broker metric
- kafka.server:name=AtMinIsrPartitionCount,type=ReplicaManager
# Partition metric
- kafka.cluster:name=AtMinIsr,type=Partition,topic={topic},partition={partition}

# New TopicCommand option
--at-min-isr-partitions
```

Proposed Changes

We will add the gauge to Partition.scala:

```
newGauge("AtMinIsr",
  new Gauge[Int] {
    def value = {
      if (isAtMinIsr) 1 else 0
    }
  },
  tags
)


...


def isAtMinIsr: Boolean = {
  leaderReplicaIfLocal match {
    case Some(leaderReplica) =>
      inSyncReplicas.size == leaderReplica.log.get.config.minInSyncReplicas
    case None =>
      false
  }
}
```

And the similar change to ReplicaManager.scala:

```
val atMinIsrPartitionCount = newGauge(
  "AtMinIsrPartitionCount",
  new Gauge[Int] {
    def value = leaderPartitionsIterator.count(_.isAtMinIsr)
  }
)
```

And TopicCommand.scala:

```
private val reportAtMinIsrPartitionsOpt = parser.accepts("at-min-isr-partitions",
  "if set when describing topics, only show partitions whose isr count is equal to the configured minimum. Not
supported with the --zookeeper option.")

private def hasAtMinIsrPartitions(partitionDescription: PartitionDescription) = {
  partitionDescription.isr.size == partitionDescription.minIsrCount
}

private def shouldPrintAtMinIsrPartitions(partitionDescription: PartitionDescription) = {
  opts.reportAtMinIsrPartitions && hasAtMinIsrPartitions(partitionDescription)
}

# Some other minor changes omitted
```

Compatibility, Deprecation, and Migration Plan

The new TopicCommand option requires use of AdminClient so it will not be available with the --zookeeper option.

Rejected Alternatives

None so far.