KIP-1018: Introduce max remote fetch timeout config for DelayedRemoteFetch requests

Status

Current state: "Under Discussion"

Discussion thread: *here*

JIRA: *KAFKA-15776*

Motivation

When reading from remote storage, we are reusing the fetch.max.wait.ms config as a delay timeout for DelayedRemoteFetchPurgatory. fetch.max.wait.ms purpose is to wait for the given amount of time when there is no local data available to serve the FETCH request.

Using the same timeout in the DelayedRemoteFetchPurgatory can confuse the user on how to configure the optimal value for each purpose. Moreover, the config is of **LOW** importance and most of the users won't configure it and use the default value of 500 ms. Having the delay timeout of 500 ms in DelayedRemoteFetchPurgatory can lead to higher number of expired delayed remote fetch requests when the remote storage have any degradation.

Public Interfaces

A new dynamic broker configuration: `fetch.remote.max.wait.ms` will be added and the delayed remote fetch purgatory will wait up to this timeout to fetch the data from the remote storage.

The consumer config **`fetch.max.wait.ms`** document will be updated to denote that it applies only to the local storage:

fetch.max.wait.ms
`The maximum amount of time the server will block before answering the fetch request when it is reading near to the tail of the partition (high-watermark) and there isn't sufficient data to immediately satisfy the requirement given by fetch.min.bytes`

Proposed Changes

Remote storage read latencies are non-deterministic. Suppose the user configures 20 remote-log-reader threads, and it takes 100 ms to serve one request. When a backfill job runs and reads data from the head of the log for multiple partitions (let's say 1000), the remote-fetch requests get queued, potentially exceeding the default timeout of 500 ms. Additionally, the time taken to serve the P99 remote storage fetch requests can extend into seconds. We propose introducing a new timeout parameter, `fetch.remote.max.wait.ms`, to offer users the option to configure the timeout based on their workload.

Under the current behavior, when the remote-fetch request times out, the client receives an empty response and retries by sending the FETCH request for the same fetch-offset. This process adds further pressure to the remote storage, as it involves fetching the same data and interrupting the thread before completion.

fetch.remote.max.wait.ms:

This parameter represents the maximum time the server will block before responding to the remote fetch request. Note that this timeout should be set to a value less than `request.timeout.ms`; otherwise, the requests will time out on the client side. The default value is configured to be 500 ms.

Setting the default value to 500 ms ensures that the client will always receive a response within 500 ms to prevent any surprises on the client side. It's important to note that there is no guarantee that the FETCH request will always be served within 500 ms. The time taken to serve the FETCH request can surpass the `fetch.max.wait.ms` due to factors such as a slow hard disk, sector errors in the disk, and so on.

Why does the configuration need to be dynamic?

By maintaining the flexibility to update this configuration dynamically, operators have more control over when to increase the timeout in case of any remote storage degradation. If there is enough remote data (`max.partition.fetch.bytes`) to respond, the remote-fetch request will be returned immediately.

Compatibility, Deprecation, and Migration Plan

- *What impact (if any) will there be on existing users?*
  No, there is no impact to the existing users as we are keeping the maintaining the same timeout
- *If we are changing behavior how will we phase out the older behavior? No*
- *If we need special migration tools, describe them here. No*
- *When will we remove the existing behavior?*

Test Plan

- The changes are covered with the existing unit and integration tests.

Rejected Alternatives

If there are alternative ways of accomplishing the same thing, what were they? The purpose of this section is to motivate why the design is the way it is and not some other way.

1. The consumer can update the existing `fetch.max.wait.ms` config to extend the timeout. This config is used when there is no enough local data to respond. We should have a separate config for each use-case.
2. Make the `fetch.remote.max.wait.ms` as a consumer config instead of broker config, the tiered storage (KIP-405) is designed such that the clients are agnostic whether they are reading from broker's local disk (or) remote storage and it's not possible to support the older clients.
3. Instead of fixed timeout, we could consider having a backoff configs and increase it up to the broker request timeout. Since, this timeout is applicable for all the FETCH requests and we don't know when to revert it back to the old/default value, maintaining this config as dynamic to give more control to the operator.