Single Reader/Writer

This wiki describes a proposal for implementing single reader/writer guarantees in Kafka.

Use Cases

The idea behind this feature is to handle cases where there is supposed to be a single reader or writer for a given partition at any time. This could easily be ensured by just having one process. However for high-availability you often want multiple replicas, only one of which is active at any given time. However failure detection is always imperfect so the possibility exists that a process may seem to be dead, causing us to allow another process to write, but it may not be dead, violating our single-writing (or reading) restriction.

Here are some specific cases we know of that would benefit from this:

- You are building a partitioned data system. Each partition has multiple replicas. There is a single master replica. You want to use Kafka as a commit log for this system.
- You have a set of processes responsible for reading data out of some external system (a database, say) and publishing to Kafka. You want to retain a total order when reading from the database so you need a single reader process, but you also want high availability so if the reader fails you want another instance to take over.
- You have multiple Kafka readers, each assigned partitions. You want to ensure that it is never the case that two readers both think they own a partition at the same time.
- Samza jobs write to a journal to make their local state highly available. It is assumed that each task is the single writer for it's journal partition. If a process dies (or seems to have died), YARN will restart that process elsewhere doing the same work.

Each of these cases have the possibility that a GC pause or other transient failure could cause a false positive in the failure detection.

The Feature

Although it is a bit hard for a group of processes to ensure that there can only be one reader or writer among them, it is actually pretty easy for the server to do this. The proposed mechanism is a new "own" API:

```
  own_request group [resource generation]
```

with a corresponding response

```
  own_response group [resource generation]
```

Issuing this request will attempt to claim ownership of a given set of topic/partitions. For our use case resources will always be topic-partition pairs, but they could be any arbitrary string.

The server will keep an optional group, and set of owned resources associated with each connection as well as a hash table of (group, resource) => (owner, generation). When a fence request is made the server will check the current connections to see if there is already anyone who has registered that resource for that group:

- If there is no such owner then the connection of the requesting process is made the owner and we return generation 1
- If the existing owner has a higher generation the request fails, returning an error_code
- If the existing owner has a an equal or lower generation the existing owner's connection is severed

Generation 0 will have special meaning. It will always succeed in claiming the resource, and will reset the generation.