**Non-Exclusive Leader Emergence in Highly Available Systems**

Emil Koutanov

**Abstract**

The Non-Exclusive Leader Emergence (NELE) protocol provides a mapping from a set of work roles to one or more assignees responsible for fulfilling each role. Its utility is in environments where it’s imperative to maintain near-continuous system availability at the expense of occasional work duplication, and where the duplication of work, while generally undesirable, does not lead to an incorrect outcome. NELE is a simple unimodal protocol that is relatively straightforward to implement, building on a shared ledger service that’s capable of atomic partition assignment, such as Kafka and Kinesis. The latter makes NELE particularly attractive for use in Cloud-based computing environments.

Published in <https://github.com/obsidiandynamics/nele> under a BSD (3-clause) license.

**Overview**

This text describes the Non-Exclusive Leader Emergence (NELE, pronounced 'Nelly') protocol built on top of a shared, partitioned ledger capable of atomic partition balancing (such as Apache Kafka or Amazon Kinesis). The protocol yields a non-exclusive leader in a group of contending processes for one of a number of notional roles, such that each role has at least one leader assigned to it. The number of roles is dynamic, as is the number of contending processes. This protocol is useful in scenarios where —

* There are a number of roles that need fulfilling, and it's desirable to share this load among several processes (likely deployed on different hosts);
* While it's undesirable that a role is simultaneously filled by two processes at any point in time, the system will continue to function correctly. In other words, this may create duplication of work but cannot cause harm (the *safety* property);
* Availability of the system is imperative; it's required that at least one leader is assigned to a role at all times so that the system as a whole remains operational and progress is made on every role (the *liveness* property);
* The number of processes and roles is fully dynamic, and may vary on an *ad hoc* basis. Processes and roles may be added and removed without reconfiguration of the group or downtime. This accommodates rolling deployments, scaling groups, and so forth;
* The use of a dedicated Group Membership Service (GMS) is deemed unviable, and where an alternate primitive is sought. Perhaps the system is deployed in an environment where a robust GMS is not natively available, but other capabilities that may internally utilise a GMS may exist. Kinesis in AWS is one such example.

**Note**: The term *emergence* is used in favour of *election* to communicate the notion of partial autonomy in the decision-making process. Under NELE, leaders aren't chosen directly, but through other phenomena that are observable by the affected group members, allowing them to infer that new leadership is in force. The protocol is also eventually consistent, in that although members of the group may possess different views, these views will invariably converge. This is contrary to a conventional GMS, where leadership election is the direct responsibility of the GMS, and is communicated to the affected parties through view changes.

**Practical implications**

Consider a system for dissemination of financial pricing data to multiple subscribers. If this is a time-critical service, then it's imperative that prices are always provided in a timely manner and the downtime of any price provider is to be avoided. A traditional high availability (HA) setup with fail-over provides continuity of price data; however, gaps during the fail-over process will be observed. A simple solution is to introduce active-active redundancy; however, this leads to duplication of pricing information.

In the above scenario, a non-exclusive leader solves the availability problem without introducing excessive data duplication. (In fact, duplication may only be observable during a leadership transition.) Furthermore, non-exclusive leadership may be used as a load-balancing technique, such that one leader is chosen for each of the independent financial instruments for which a stream of pricing data exists. So a distributed cluster of processes may share the load of publishing price streams without overlapping under normal conditions; no minority subset of processes is responsible for publishing of the majority of data streams.

**Protocol definition**

A centrally-arbitrated topic *C* is established with set of partitions *M*. (Assuming *C* is hosted by Kafka or Kinesis.) A set of discrete roles *R* is available for assignment, and a set of processes *P* contend for assignment of roles in *R*. The number of elements in the set *M* may vary from that of *R*which, in turn, may vary from *P*.

Each process in *P* continually publishes a message on all partitions in *M* (each successive message is broadcast a few seconds apart). The message has no key or value; the producing process explicitly specifies the partition number for each published message. As each process in *P* publishes a message to *M*, then each partition in the set *M* is continually subjected to messages from each process. Corollary to this, for as long as at least one process in *P* remains operational, there will be at least one message continually published in each partition in *M*. Crucial to the protocol is that no partition may 'dry up'.

Each process *p* in *P* subscribes to *C* within a common, predefined consumer group. As per Kafka's partition assignment rules, a partition will be assigned to at most one consumer. Multiple partitions may be assigned to a single consumer, and this number may vary slightly from consumer to consumer. Note — this is a fundamental assumption of NELE, requiring a broker that is capable of arbitrating partition assignments.

Each process *p* in *P*, now being a consumer of *C*, will maintain a vector *V* of size identical to that of *M*, with each vector element corresponding to a partition in *M*, and initialised to zero. *V* is sized during initialisation of *p*, by querying the brokers of *M* to determine the number of partitions in *M*, which will remain a constant. (As opposed to elements in *P* and *R* which may vary dynamically.) This implies that *M* may not be expanded while a group is in operation.

Upon receipt of a message *m* from *C*, *p* will assign the current machine time as observed by *p* to the vector element at the index corresponding to *m*'s partition index.

Assuming no subsequent partition reassignments have occurred, each *p*'s vector comprises a combination of zero and non-zero values, where zero values denote partitions that haven't been assigned to *p*, and non-zero values correspond to partitions that have been assigned to *p* at least once in the lifetime of *p*. If the timestamp at any of vector element *i* is *current* — in other words, it is more recent than some predefined constant threshold *T* that lags the current time — then *p* is a leader for *Mi*. If partition assignment for *Mi* is altered (for example, if *p* is partitioned from the brokers of *M*, or a timeout occurs), then *Vi* will cease incrementing and will eventually be lapsed by *T*. At this point, *p* must no longer assume that it's the leader for *Mi*.

Ownership of *Mi* still requires a translation to a role assignment, as the number of roles in *R* may vary from the number of partitions in *M*, and in fact, may do so dynamically without prior notice. To determine whether *p* is a leader for role *Rj*, *p* will compute *k = j mod size(M)* and check whether *p*is a leader for *Mk* through inspection of its local *Vk* value.

Where *size(M) > size(R)*, ownership of a higher numbered partition in *M* does not necessarily correspond to a role in *R* — the mapping from *R* to *M* is injective. If *size(M) < size(R)*, ownership of a partition in *M* corresponds to (potentially) multiple roles in *R*, i.e. *R* → *M* is surjective. And finally, if *size(M) = size(R)*, the relationship is purely bijective. Hence the use of the modulo operation to remap the dynamic extent of *R* for alignment with *M*, guaranteeing totality of *R* → *M*.

**Note**: Without the modulo reduction, *R* → *M* will be partial when *size(M) < size(R)*, resulting in an indefinitely dormant role and violating the *liveness* property of the protocol.

Under non-exclusive leadership, the value of *T* is chosen such that *T* is greater than the partition reassignment threshold of the broker. In Kafka, this is given by the property session.timeout.ms on the consumer, which is 10 seconds by default — so *T* could be 30 seconds, allowing for up 20 seconds of overlap between successive leadership transitions. In other words, if the partition assignment is withdrawn from an existing leader, it may presume for a further 20 seconds that it is still leading, allowing for the emerging leader to take over. In that time frame, one or more roles may be fulfilled concurrently by both leaders — which is acceptable *a priori*. (In practice, 10 seconds is too long for HA systems that approach continuous availability; smaller values such as 100 milliseconds are more suitable.)

There is no hard relationship between the sizing of *M*, *R* and *P*; however, the following guidelines should be considered:

* *R* should be at least one in size, as otherwise there are no assignable roles.
* *M* should be sized equal to or less than *R*, so as to avoid processes that have no actual *role* assignments in spite of owning one or more partitions (for high numbered partitions). When using Kafka, this avoids the problem when partition.assignment.strategy is set to range, which happens to be the default. To that point, it is recommended that the partition.assignment.strategy property on the broker is set to roundrobin, so as to avoid injective *R* → *M* mappings that are extremely asymmetric.
* *M* should be sized approximately equal to the steady state (anticipated) size of *P*, notwithstanding the fact that *P* is determined dynamically, through the occasional addition and removal of deployed processes. When the size of *M* approaches the size of *P*, the assignment load is shared evenly among the constituents of *P*.

It is also recommended that the session.timeout.ms property on the consumer is set to a very low value, such as 100 for rapid consumer failure detection and sub-second rebalancing. This requires setting of group.min.session.timeout.ms on the broker to 100 or lower, as the default value is 6000. The heartbeat.interval.ms property on the consumer should be set to sufficiently small value, such as 10.

**Further considerations**

**Exclusivity of role assignment**

It can also be shown that the non-exclusivity property can be turned into one of exclusivity through a straightforward adjustment of the protocol. In other words, the at-least-one leader assignment can be turned into an at-most-one. This would be done in systems where non-exclusivity cannot be tolerated.

The non-exclusivity property is directly controlled by the liberal selection of the constant *T*, being significantly greater than the partition reassignment threshold. If, on the other hand, *T* is chosen conservatively, such that T is significantly less than the reassignment threshold, then the currently assigned leader will expire prior to the assignment of its successor, leaving a gap between successive assignments.

**Topic reuse for independent role-process assignments**

By using a different consumer group ID, the same set of partitions *M* can be exploited to assign a different set of roles *R'* to a different set of processes *P'*. In other words, there is no compelling need to establish a new topic for a different set of roles.

**Use in continuously available systems**

By the term *non-exclusive leader* it is meant that at least one leader may be assigned; it shouldn't be taken that the assigned leader is actually functioning, network-reachable and is able to fulfil its role at all times. As such, single-group NELE cannot be used directly within a setting of strictly continuous availability, where at least two leaders are *always* required. Of course, the thresholds used in NELE can be tuned such that the transition time between leaders is minimal, at the expense of work duplication. However, while being arbitrarily responsive (near-continuous), this doesn't formally satisfy continuous availability guarantees, which assume zero downtime under a finite set of assumptions (for example, at most one component failure is tolerated).

In a continuously available system, two or more disjoint (non-overlapping) NELE process groups *P1* and *P2* (through to *PN*, if necessary) may be used concurrently on the same set *M* (or a different set *M'*, hosted on an independent set of brokers) and a common *R*, such that any *Ri* would be assigned to a member in *P1* and *P2*, such that there will be at least two leaders for any *Ri* at any point in time -- one from each set. Depending on the value of *T*, there may be three leaders during a transition event in any of the groups *P1* or *P2*, assuming that process failures in *P1* and *P2* are uncorrelated.

Furthermore, it is prudent to keep the process sets *P1* and *P2* not only disjoint, but also deployed on separate hosts, such that the failure of a process in *P1* will not correlate to a failure in *P2*, where both failed processes may happen to share role assignments.