(Partition Map) Exchange - under the hood

Notes:

- Following notation is used: words written in italics and without spacing mean class names without package name, for example, *GridCacheMapEntr*

Contents:

(Partition Map) Exchange

PME - is process of exchange partition information between nodes. Process goal is to setup actual state of partitions for all cluster nodes.

Triggers

Events which causes exchange

- Node Join (*EVT_NODE_JOINED*) - new node discovered and joined topology (exchange is done after node is included into ring)
- Node Left (*EVT_NODE_LEFT*) - correct shutdown with call ignite.close
- Node Failed (*EVT_NODE_FAILED*) - detected unresponsive node, probably crashed and is considered failed
- Custom events with flag exchange Needed
- Start (dynamic) cache / stop cache

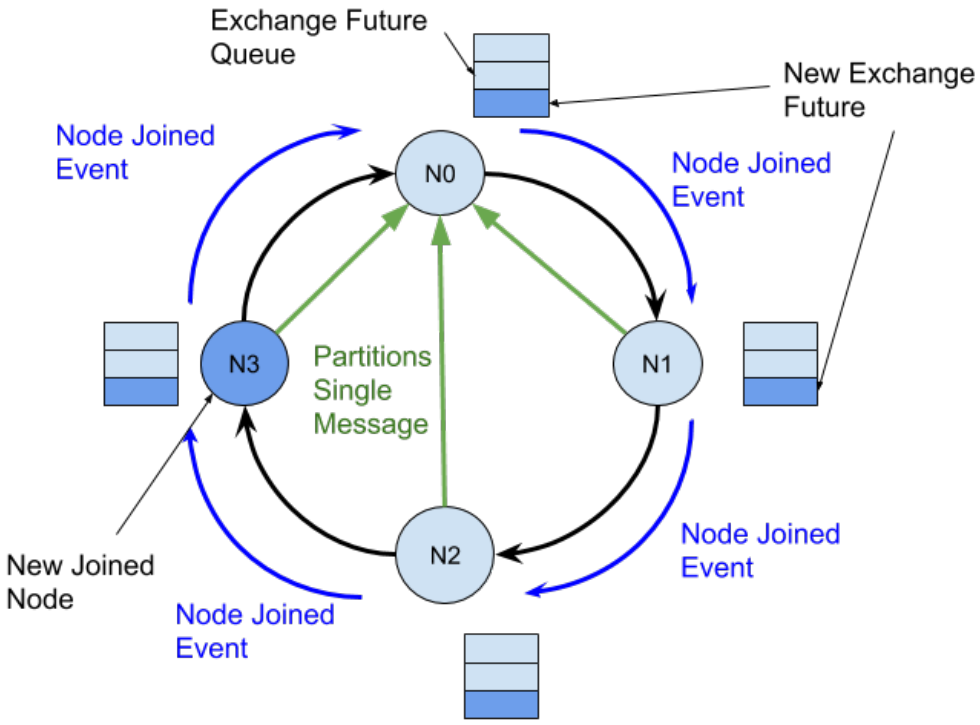
Exchange process

Partition Map Exchage process is following

- Non coordinator nodes sends its local state (*GridDhtPartitionsSingleMessage*)
- Coordinator merges local partition information. Coordinator builds partition full map
- Coordinator sends full map to other nodes (*GridDhtPartitionsFullMessage*)

Phase 1. Accumulation of partition state

Cluster members change causes major topology update (e.g. 5.06.0), cache creation or destroy causes minor version update (e.g. 6.06.1).



Initialization of exchange means adding future *GridDhtPartitionsExchangeFuture* to queue on each node.

Put to this queue (*GridCachePartitionExchangeManager.ExchangeWorker*#futQ) is done from discovery thread.

'Exhange worker' thread manages this queue. Using only one exhange worker thread per node provides strict processing order for futures.

Step 1) First of all exhange worker thread runs init() on this exchange future. This method sends single map (*GridDhtPartitionsSingleMessage*). Signle map is sent using communication SPI (in peer2peer manner).

This signle message contains partitions map for each cache (*GridDhtPartitionMap*). It contains owners of each paritition for each cache group (cache group id is mapped to Partitions Map). Following is example from log for 2 caches:

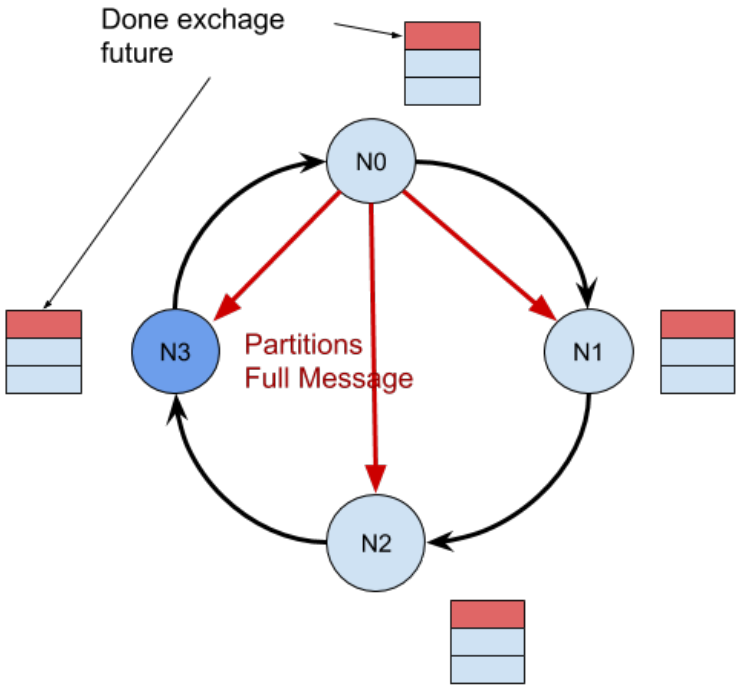
```
msg=GridDhtPartitionsSingleMessage
[parts=
{1544803905=GridDhtPartitionMap [moving=0, top=AffinityTopologyVersion [topVer=9, minorTopVer=3], updateSeq=2,
size=406],
-2100569601=GridDhtPartitionMap [moving=0, top=AffinityTopologyVersion [topVer=9, minorTopVer=3], updateSeq=161,
size=100]}
```

Phase 2. Building and sharing full state

Step 2) Coordinator maintains list of nodes to wait response from (N1, N2, N3 for example at pic.1).

According to single maps received and to its own state coordinator build full map.

When all expected responses were received coordinator sends full map to other nodes.



Following is example of logging of 'Sending full partition map' for 2 caches:

```
msg=GridDhtPartitionsFullMessage [
 parts={
  1544803905=GridDhtPartitionFullMap [nodeId=1c6a3487-0ad2-4dc1-a69d-7cd84db00000, nodeOrder=1, updateSeq=6,
size=5],

  -2100569601=GridDhtPartitionFullMap [nodeId=1c6a3487-0ad2-4dc1-a69d-7cd84db00000, nodeOrder=1, updateSeq=102,
size=5]},

 topVer=AffinityTopologyVersion [topVer=9, minorTopVer=3]
```

Partition state in full and signle maps are encoded using bit sets for compression.

On receiving full map for current exchange, nodes calls onDone() for this future. Listeners for full map are called (order is no guaranteed).

Example of exchange and rebalancing for node join event

Preconditions

Example of possible partitions state for 3 nodes, 3 partition, 1 backup per partition

```
Node 1: Primary P1 - Owning
        Backup P2  - Owning
Node 2: Primary P2 - Owning
        Backup P3  - Owning
Node 3: Backup P1  - Owning
        Primary P3 - Owning
```

Each node knows following:

- local partitions info - it is always up to date,
- all cluster nodes partitions state - may have lag behind cluster state

Server node with lowest order (an oldest alive node in topology) is coordinator (there is always one such node) - Usually 'crd' in code

Let's suppose now new node is joining (it would be Node 4).

All other nodes know state of partitions that coordinator had observed some time ago.

Partition mapping is mapping of following: UUID -> partition ID -> State

PME and rebalancing

Step 1. Discovery event triggers exchange

Topology version is incremented

Affinity generates new partition assignment

For example, Partition 3 Backup is now assigned to new node 4 (to be moved to N4)

Step 2. Node 4: Partition 3 Backup is created at new node at state Moving:

```
Node 2: Primary P2 -  Owning
        Backup P3 -  Owning
...
Node 4:
        Backup P3 -  Moving
```

Node 2 does not throw data and does not change partition state even it is not owning partition by affinity on new topology version.

Step 3. Node 4 starts to issue demand requests for cache data to any node which have partition data.

The full-map exchange most likely will be completed before node 4 rebalances all the data it will own (with node 4 partition 3 state - Moving),

Step 4. When all data is loaded (last key is mapped) Node 4 locally changes state to Owning (other nodes think it is Moving).

Node schedules a background cluster notification.

Step 5. Node 4 after some delay (timer) sends single message to Coordinator. This message will have absent/'*null*' version of exchange.

Step 6. Coordinator sends updated full map to other nodes.

Step 7. Node 2: observes that

- in current topology there are 3 owners of partition,
- affinity says should be 2
- and current node is not affinity node for the partition.

Partition can be safely removed.

Partition state is changed to Renting locally (other nodes think it is Owning).

Step 8. Node 2 after some delay (timer) sends single message to crd.

Step 9. Coordinator sends updated full map to other nodes.

All nodes eventually get renting state for P3 backup for Node 4.

SingleMessage with *exchId*=null is sent when a node updates local partitions' state and schedules a background cluster notification.

In contrast, when a partition map exchange happends, it is completed with *exchId* != null.

If more topology updates occurred, this causes more cluster nodes will be responsible for same partition at particular moment.

Clearing of cache partition data immediately is not possible because of SQL indexes. These indexes covers all node's partitions globally. Elements removed one by one. Later state will be Evicted.

Exchange types

- synchronous: Come from discovery events (real & custom): have to wait transactions to finish
- asynchronous, does not slowdown operations

Synchronous Exchange

DiscoveryEvent is sent to cluster using ring (*DiscoverySpi*).

Synchronous exchange is required for primary partition rebalancing to new node.

Let's suppose for a second, such partition map exchange is not synchronous: There is possible case that 2 primaries exists in Owning state. New node changed state locally, old node does not received update yet. In that case lock requests may be sent to different nodes, it could brake ACID.

As result, primary migration requires global synchronous exchange. It should not be running in the same time with transactions.

For protection there is Topology ReadWrite Lock

- transactions & atomic cache updates - acquire read lock - N locks supported in the same time
- exchange acquire write lock - 1 lock excluding transactions is possible

Using ReadWrite lock gives semantic N transactions, or 1 exchange.

Late Affinity Assignment

Optimisation, Centralized affinity, flag - true by default 2.0+, the only possible option since 2.1+.

Affinity assigns primary partition (to be migrated) to new node. But instead we create temporary backup at this new node.

When temporary backup loads all actual data it becomes primary.

Additional synthetic Exchange will be issued.