Redis Architecture

http://redis.io/presentation/Redis_Cluster.pdf
All nodes are directly connected with a service channel.
TCP baseport+4000, example 6379 -> 10379.
Node to Node protocol is binary, optimized for bandwidth and speed.
Clients talk to nodes as usually, using ascii protocol, with minor additions.
Nodes don't proxy queries.

Hash slots
keyspace is divided into 4096 hash slots.
Different nodes will hold a subset of hash slots.
Nodes are all connected and functionally equivalent, but
actually there are two kind of nodes: slave and master nodes:

there are two replicas per every master node, so
up to two random nodes can go down without issues.
Working with two nodes down is guaranteed, but in the best case the cluster will continue to work as long as there is at least one node for every hash slot.

Every key only exists in a single instance, plus N replicas that will never receive writes. So there is no merge, nor application-side inconsistency resolution.
The price to pay is not resisting to net splits that are bigger than replicas-per-hashslot nodes down.
Master and Slave nodes use the Redis Replication you already know.

Every physical server will usually hold multiple nodes, both slaves and masters, but the redis-trib cluster manager program will try to allocate slaves and masters so that the replicas are in different physical servers

Client requests - dummy client
1. Client => A: GET foo
2. A => Client: -MOVED 8 192.168.5.21:6391
3. Client => B: GET foo
4. B => Client: "bar"
-MOVED 8 ... this error means that hash slot 8 is located at the specified IP/port, and the client should reissue the query there.

Client requests - smart client
1. Client => A: CLUSTER HINTS
2. A => Client: ... a map of hash slots -> nodes
3. Client => B: GET foo
4. B => Client: "bar"

Client requests
Dummy, single-connection clients, will work with minimal modifications to existing client code base. Just try a random node among a list, then reissue the query if needed.

Smart clients will take persistent connections to many nodes, will cache hashslot -> node info, and will update the table when they receive a -MOVED error.

This schema is always horizontally scalable, and low latency if the clients are smart.
Especially in large clusters where clients will try to have many persistent connections to multiple nodes, the Redis client object should be shared.

Re-sharding - moving data
All the new keys for slot 7 will be created / updated in D.
All the old keys in C will be moved to D by redis-trib using the MIGRATE command.
MIGRATE is an atomic command, it will transfer a key from C to D, and will remove the key in C when we get the OK from D. So no race is possible.
p.s. MIGRATE is an exported command

Re-sharding with failing nodes
Nodes can fail while resharding. It's slave promotion as usually.
The redis-trib utility is executed by the sysadmin. Will exit and warn when something is not ok as will check the cluster config continuously while resharding.

Fault tolerance
All nodes continuously ping other nodes...
A node marks another node as possibly failing when there is a timeout longer than N seconds.
Every PING and PONG packet contain a gossip section: information about other nodes idle times, from the point of view of the sending node.

Fault tolerance - failing nodes
A guesses B is failing, as the latest PING request timed out.
A will not take any action without any other hint.
C sends a PONG to A, with the gossip section containing information about B: C also thinks B is failing.
At this point A marks B as failed, and notifies the information to all the other nodes in the cluster, that will mark the node as failing.
If B will ever return back, the first time he'll ping any node of the cluster, it will be notified to shut down ASAP, as intermitting clients are not good for the clients.

Redis-trib - the Redis Cluster Manager
It is used to setup a new cluster, once you start N blank nodes.
it is used to check if the cluster is consistent. And to fix it if the cluster can't continue, as there are hash slots without a single node.
It is used to add new nodes to the cluster, either as slaves of an already existing master node, or as blank nodes where we can re-shard a few hash slots to lower other nodes load.

Ping/Pong packets contain enough information for the
cluster to restart after graceful stop. But the sysadmin can use CLUSTER MEET command to make sure nodes will engage if IP changed and so forth.
Every node has a unique ID, and a cluster config file. Everytime the config changes the cluster config file is saved.
The cluster config file can't be edited by humans.
The node ID never changes for a given node.
http://engineering.bloomreach.com/the-evolution-of-fault-tolerant-redis-cluster/
http://www.yzuzun.com/2015/04/some-architectural-design-concepts-for-redis/