Huge Distributed Cache Design Documentation

1. Overview, Requirements, and Assumptions

A. Functional Requirements:

- Key-Value Data Storage: In-memory storage for fast access using multiple data structures.
- Data Expiration and Eviction: Support TTLs and eviction policies (LRU, LFU, etc.).
- Atomic Operations: Execute commands atomically (INCR, MULTI/EXEC) for consistency.
- Pub/Sub (Optional): Support real-time messaging between clients.
- Persistence (Optional): Snapshotting and append-only logging for durability.

B. Nonfunctional Requirements:

- Low Latency and High Throughput: Operations should execute in microseconds to milliseconds.
- Scalability: Handle billions of operations per day across thousands of nodes.
- Fault Tolerance: Maintain service availability with replication and automatic failover.
- Global Distribution: Deploy across multiple regions for low latency worldwide.
- Monitoring and Manageability: Provide dashboards and logging for operational insights.

C. Assumptions:

- The cache is used primarily as a front for high-traffic applications.
- Data is partitioned into hash slots and stored in memory.
- Clients use a custom protocol over TCP with TLS for security.
- The system can optionally persist data for durability and recovery.

2. High-Level Architecture and Component Responsibilities

A. Client Interface:

- Client libraries in various languages communicate with the cache using a custom TCP protocol over TLS.

B. API Gateway / Smart Client Routing:

- Clients determine the appropriate node responsible for a given key based on a consistent hashing algorithm.
- Optionally, a proxy layer or smart client handles request routing.

C. Cache Nodes (Master and Replicas):

- Master Nodes store key-value pairs in assigned hash slots.
- Replica Nodes provide read redundancy and take over in case a master fails.

D. Cluster Management and Coordination:

- A Cluster Manager (similar to Redis Sentinel) monitors node health, manages slot reallocation, and orchestrates failover.

E. Persistence (Optional):

- Snapshot and Append-Only File (AOF) systems allow data to be saved periodically for recovery.

F. Monitoring and Logging:

- Tools are integrated to monitor memory usage, request latency, node health, and throughput.

3. Detailed Workflow

A. Client Request Flow:

1. The client computes a hash for the key and determines the corresponding slot.
2. The request is directed to the master node responsible for that slot via DNS routing or a smart client.
3. The master node executes the requested command atomically and returns the result.

B. Data Partitioning and Replication:

1. The full key space is divided into slots (e.g., 16384 slots).
2. Each master node is assigned a subset of slots; replicas mirror these keys asynchronously.

C. Failover and Reconfiguration:

1. The Cluster Manager detects a master failure and promotes a replica to master.
2. Clients are updated with the new slot mapping to ensure continued operation.

D. Optional Persistence:

1. Periodic snapshots and an append-only log capture the current state.
2. On startup, nodes can reload data from snapshots/AOF to recover previous state.

4. Scalability, Fault Tolerance, and Global Distribution

A. Horizontal Scalability:

- The key space is sharded across thousands of nodes; each node handles a fraction of operations.
- Autoscaling policies automatically spin up new nodes as load increases.

B. Fault Tolerance:

- Each master node has at least one replica; automatic failover ensures high availability.
- The Cluster Manager continuously monitors health and reassigns slots as necessary.

C. Global Distribution:

- The cache is deployed in multiple regions; global DNS routing directs clients to the nearest region.

- Regional clusters minimize latency and enhance performance for a global user base.

5. Protocols, Security, and External Integrations

A. Communication Protocols:

- A custom binary or text-based protocol (similar to Redis protocol) is used over TCP.
- TLS encrypts all client-to-server communications to ensure secure data transfer.

B. Interservice Communication:

- gRPC or REST over secured TCP channels (with mutual TLS) coordinate between cluster management nodes and backup services.

C. Security:

- Authentication (if required) and access control lists (ACLs) prevent unauthorized access.
- Sensitive data can be encrypted at rest in optional persistence layers.

D. Monitoring and Management:

- Integration with Prometheus, Grafana, and custom dashboards for real-time cluster monitoring.
- Administrative tools allow cluster reconfiguration and node diagnostics.

6. Final Thoughts

This design for a huge distributed caching systemmodeled after Redisoffers a robust solution for highvolume, lowlatency data storage. Key advantages include:

- Extremely fast in-memory operations that serve as a critical component for high-traffic applications.
- Horizontal scalability via sharding and consistent hashing that distributes the key space across thousands of nodes.
- High availability through master-replica replication and automatic failover orchestrated by a cluster management service.
- Global distribution capabilities that reduce latency by serving clients from the nearest region.
- Flexible design that can optionally support persistence for durability and recovery.

This architectural framework provides a solid foundation for building a scalable, efficient, and secure distributed cache that can support modern, high-performance applications at global scale.