Contents

Less Common Questions

Design a system with capped storage

When storage is limited, design for data retention and efficiency: - **Log Compaction:** Periodically remove obsolete or duplicate records (e.g., Kafka log compaction). - **Data Expiration:** Apply TTL (time-to-live) policies to automatically delete old data. - **Aggregation:** Store only summaries or aggregates (e.g., daily counts instead of raw events). - **Deduplication:** Remove redundant events before storage. - **Monitoring:** Track storage usage and alert before reaching capacity.

Scale the Like button

Design for high write throughput and real-time feedback: - **Batch Writes:** Queue and batch like events to reduce DB write load. - **Caching:** Store like counts in an in-memory cache (e.g., Redis) and periodically sync to the database. - **Approximate Counters:** Use probabilistic data structures (e.g., HyperLogLog) for non-critical real-time counts. - **Backfill:** Reconcile and backfill counts during off-peak hours. - **Hotspot Mitigation:** Shard by post ID to distribute load.

Distributed cache

Build a scalable, resilient cache layer: - **Consistent Hashing:** Evenly distribute keys and allow easy node addition/removal. - **Replication:** Replicate data across nodes to prevent data loss and hotspots. - **Eviction Policies:** Use LRU (Least Recently Used) and TTL (Time-to-Live) to manage memory. - **Monitoring:** Track hit/miss ratios and latency. - **Invalidation:** Invalidate or update cache entries on data changes to maintain consistency.

Groups for poor connectivity

Support users with intermittent or slow connections: - **Local Storage:** Cache recent threads and messages on device for offline access. - **Write-Ahead Logs:** Log outbound actions locally for retry when connectivity returns. - **Batch Sync:** Sync changes in batches to reduce network overhead. - **Conflict Resolution:** Use timestamps or version vectors to resolve sync conflicts. - **User Experience:** Indicate offline/online status and sync progress in the UI.

Archive all Facebook posts

Efficiently move old data to cold storage: - **Archival Triggers:** Set TTL or user-driven triggers for archiving. - **Cold Storage:** Move data to cost-effective storage (e.g., S3, Glacier) with metadata preserved. - **Restore on Demand:** Queue and process restore requests when users access archived posts. - **Indexing:** Maintain lightweight indexes for fast lookup. - **Monitoring:** Track archive health, access logs, and restore success rates.

SLA-based job queue

Ensure jobs meet deadlines and SLAs: - **Priority Queuing:** Use weighted fair queues or priority queues based on job deadlines. - **Deduplication:** Remove duplicate jobs before processing. - **Job Stealing/Bumping:** Allow workers to steal or reprioritize jobs to meet SLAs. - **Monitoring:** Track job latency, deadline misses, and queue length. - **Backpressure:** Apply rate limits or reject jobs when overloaded.

Self-healing service

Design for resilience and automatic recovery: - **Health Checks:** Continuously monitor service health and dependencies. - **Auto-Scaling:** Scale up/down based on load and health. - **Canary Rollouts:** Deploy changes to a subset of instances and monitor for issues. - **Auto-Revert:** Automatically roll back on error spikes or failed health checks. - **Chaos Testing:** Regularly inject failures to validate recovery mechanisms. - **Alerting:** Notify operators only when automated recovery fails.