

**Choosing the Right Database for a Fantasy-Game Platform**

**Problem Statement**

A new **Fantasy Game** platform must store large volumes of user profiles, teams, transfers and real-time gameplay activity while delivering low-latency APIs under heavy concurrency. Three distributed databases—**Cassandra**, **ScyllaDB** and **CockroachDB**—were evaluated through full Proof-of-Concept (POC) test suites covering five critical APIs (userLogin, getUserProfile, saveTeam, getUserTeams, transferTeam). The goal was to identify which engine offers the best mix of performance, fault-tolerance and operational simplicity for this workload.

**Database Comparisons**

**Cassandra**

* **Why it fits**
  + Mature wide-column model maps cleanly to partition-key access patterns (*partition\_id*, *user\_bucket*) used by the game.
  + Tunable consistency (LOCAL\_ONE in tests) gives predictable sub-15 ms reads for profile and team look-ups.
  + Huge community, abundant tooling.
* **Cons**
  + JVM-based; non-trivial GC tuning at scale.
  + Hot-partition risk when write patterns are skewed.
  + Schema changes and repairs add operational toil.

**ScyllaDB**

* **Why it fits**
  + Drop-in wire-protocol compatible with Cassandra but **rewritten in C++** for shard-per-core architecture; eliminates JVM overhead.
  + Built-in shard awareness, high IOPS and low p-99 latency; handled **40 peak concurrent workers** without back-pressure.
  + Optimised batch writes reduced end-to-end saveTeam latency by ~2× over Cassandra with identical schema.
* **Cons**
  + Smaller ecosystem than Cassandra.
  + Requires careful CPU pinning and NUMA awareness for peak benefit.
  + Some advanced features (e.g., secondary indexes) still evolving.

**CockroachDB**

* **Why it fits**
  + Postgres-compatible SQL eased development; no driver change.
  + Strong consistency and automatic rebalancing across nodes—simplifies multi-region growth.
  + Native distributed transactions simplified multi-statement saveTeam + transfer logic.
* **Cons**
  + Higher average write latency (global consensus on Raft) observed—saveTeam averaged **68.9 ms** vs 15–30 ms on the others.
  + Requires careful schema and index tuning to avoid leaseholder hotspots.
  + Fewer knobs to relax consistency for extreme throughput.

**POC Parameters and APIs**

|  |  |
| --- | --- |
| Aspect | Implementation Highlights |
| Test APIs | userLogin, getUserProfile, saveTeam, getUserTeams, transferTeam implemented in Node.js for all three back-ends. |
| Sharding / Partitioning | Hash on source\_id → partition\_id (0-29) for Cockroach; partition\_id + user\_bucket for Cassandra/Scylla. |
| Concurrency | 120 k logins, 100 k–200 k reads, ~100 k writes per run; peak 40 async workers for Scylla, 15 for Cassandra, 25 for Cockroach. |
| Consistency levels | Cockroach: serialisable (default). Cassandra: LOCAL\_ONE / LOCAL\_QUORUM. Scylla: LOCAL\_QUORUM with shard awareness. |
| Schema | Wide-row tables for game users, teams and transfers; identical logical model across DBs; Cockroach used SQL transactions, others used batches. |

**Test Results Comparison**

|  |  |  |  |
| --- | --- | --- | --- |
| Metric (avg unless noted) | Cassandra | ScyllaDB | CockroachDB |
| userLogin latency | 10.5 ms | 19.9 ms | 49.9 ms |
| getUserProfile latency | 5.4 ms | 10.2 ms | 8.0 ms |
| saveTeam latency | 15.8 ms | 30.5 ms | 68.9 ms |
| getUserTeams latency | 5.8 ms | 10.4 ms | 29.9 ms |
| transferTeam latency | 16.7 ms | 30.8 ms | 27.9 ms |
| Peak throughput (ops/s) | ~420 | **525** | ~310 |
| Total operations | 358 k | **505 k** | 560 k |
| Error rate | 0% | 0% | 0.01% |
| Memory footprint (Node heap) | 31 MB | **26 MB** | 33 MB |

*All runs executed on identical three-node Docker clusters (2 vCPU, 2 GiB each).*

**Findings**

* **Fastest overall**: *Cassandra* delivered the lowest average latencies on four of five APIs thanks to lightweight LOCAL\_ONE reads and small batch writes.
* **Highest throughput & scalability**: *ScyllaDB* sustained the greatest operations-per-second and peak concurrency with zero errors, validating its shard-per-core design for write-heavy game traffic.
* **Strong consistency & operational ease**: *CockroachDB* excelled at transparent fail-over and distributed SQL, but its Raft consensus introduces higher write latencies that may affect in-game responsiveness.

**Recommendation**

For a single-region launch where **latency and throughput** outweigh strong per-row transactions, **ScyllaDB** offers the best blend of speed and fault-tolerance while keeping the Cassandra-style data model. Cassandra remains a safe fallback with broader ecosystem support. If the roadmap demands multi-region serialisable SQL without major latency sensitivity, CockroachDB becomes attractive despite slower writes.

⁂