Database Concurrency Comparison - Single Machine Performance

Database	Default Connection Limit	Concurrency Model	Locking Mechanism	Memory Per Connection	Practical Connection Limit	Key Bottlenecks	Best Use Cases
MySQL	100-151	Thread per connection	Row-level (InnoDB)	2-10MB	500-1,000	Thread overhead, lock contention	Structure data with complex relationsh
MongoDB	65,536	Thread pool with async I/O	Document-level	Lower (~1MB)	Several thousand	Storage engine, query complexity	Documen oriented data, variable schema
Firebase (Firestore)	Serverless (Google managed)	Event-driven, async	Optimistic concurrency	N/A (serverless)	1M+ (quota limited)	API rate limits	Mobile apps, rea time applicatio
PostgreSQL	100	Process per connection	MVCC	~10MB	300-500	Process management overhead	Complex transactio data integrity
Redis	65,535	Single-threaded event loop	Atomic operations	Very low	5,000-10,000	Single-threaded processing	Caching, pub/sub, simple da structures
Cassandra	No connection limit	Staged event-driven architecture	Optimistic	Low	10,000+	Hardware resources	Time-seri data, distribute workload
CockroachDB	Configurable	Distributed SQL engine	Optimistic (MVCC)	Moderate	1,000+	Network latency	Global distribute applicatio
SQLite	1 writer, many readers	Single connection	File-level	Very low	Limited by file locking	Write concurrency	Embedde applicatio local storage

Notes on Maximizing Concurrency

1. **Connection Pooling**: For SQL databases, using connection poolers like PgBouncer (PostgreSQL) or ProxySQL (MySQL) can dramatically increase effective concurrency.
2. **Hardware Considerations**:
   - RAM is typically the primary limiting factor for connection count
   - CPU cores affect query processing capacity
   - Disk I/O can bottleneck write-heavy workloads
3. **Application Architecture**:
   - Efficient connection management (pooling, reconnection)
   - Query optimization to reduce operation duration
   - Caching to reduce database hits
   - Asynchronous processing where possible
4. **Scaling Options**:
   - Vertical scaling: Larger server with more RAM/CPU
   - Read replicas: Distribute read queries
   - Sharding: Partition data across machines
   - Serverless options: Eliminate connection management

The actual concurrency capability depends significantly on your specific workload patterns, hardware configuration, and optimization efforts.