Core Architecture & Data Model

Feature	Redis Enterprise	GridGain Enterprise
Architecture	Shared-nothing, in-memory, sharded	Distributed MPP (Massively Parallel Processing) architecture
Data model	Key-value, data structures (List, Set, Hash, JSON, Stream, TimeSeries)	Key-value + full **ANSI SQL tables** + ACID transactions + collocated compute
Memory model	DRAM-based with optional SSD tiering	Off-heap memory with **durable memory (RAM+Disk)**
Data ownership	Cache-first (copy of truth); apps control sync	**In-memory data grid (primary store of record)**
Data partitioning	Hash-based sharding	Affinity-based partitioning with compute/data co-location

Transactions & Consistency

Feature	Redis Enterprise	GridGain Enterprise
Transaction support	Single-shard, optimistic multi-command transactions	**Distributed, ACID-compliant transactions** with 2PC (two-phase commit)
Locking model	None (optimistic locking only)	Pessimistic & optimistic locking, deadlock prevention
Cross-node consistency	Not supported	Fully supported with global coordination
JTA/XA support	Not supported	Supported

Persistence & Durability

Feature	Redis Enterprise	GridGain Enterprise
Persistence options	RDB snapshots + AOF logs	**Native persistence**, write-ahead logging, checkpointing
Startup behavior	Requires reload from disk	**Instant-on recovery** from durable memory
Data loss on crash	Possible unless carefully tuned	Zero data loss with journaling and checkpointing

Querying & Analytics

Feature	Redis Enterprise	GridGain Enterprise
Query language	Redis commands, custom APIs (Search/JSON/TimeSeries)	**ANSI SQL (DDL/DML), JDBC/ODBC**, full relational support
Secondary indexes	Yes, via RedisSearch	Yes, including distributed indexes
Join support	No	**Yes, distributed joins supported**
Aggregation & filtering	Yes (limited)	Yes (SQL GROUP BY, HAVING, etc.)
Analytics engine	Limited	**Distributed MPP compute**

Compute & Streaming

Feature	Redis Enterprise	GridGain Enterprise
Pub/Sub support	Yes	Yes
Streaming engine	Redis Streams	**Built-in stream processor + Kafka/Spark integration**
Server-side compute	No (uses Lua scripts or client logic)	**Yes, collocated processing (map/reduce, ML, continuous queries)**
ML integration	External models only	Integrated with Apache Ignite ML + collocated scoring

Clustering & Scalability

Feature	Redis Enterprise	GridGain Enterprise
Cluster type	Shared-nothing, shard-based	Peer-to-peer, partitioned MPP cluster
Sharding	Auto-sharding, hash-based	Affinity-aware partitioning
Replication	Active-active with CRDTs	Synchronous/asynchronous replication
Elastic scaling	Supported	Supported with automatic rebalancing

Feature	Redis Enterprise	GridGain Enterprise
Compute/data co-location	Not supported	**Fully supported** for performance gains

Security, Monitoring, Tooling

Feature	Redis Enterprise	GridGain Enterprise
Security	TLS, RBAC, auditing, compliance (SOC2, HIPAA)	TLS, RBAC, auditing, LDAP, Kerberos, fine-grained access control
Monitoring & management	Redis Insight, CLI, REST	**GridGain Control Center** for monitoring, config, snapshot management
Backup & restore	Manual/automated snapshotting	**Point-in-time recovery**, snapshot scheduling
Deployment options	Cloud, Kubernetes, hybrid	On-prem, hybrid, Kubernetes, cloud-native

Use Cases (Investment Banking & Markets)

Application Area	Redis Enterprise	GridGain Enterprise
Market data cache	Sub-ms pricing cache, tick data lookup	Used as upstream cache, but better suited for risk calcs
Trade matching / lifecycle	Not ACID-safe for cross-shard trade ops	**Yes – full ledger handling with ACID**
Fraud detection	Real-time feature store + ML model scoring	Inline model execution with stream + collocated compute
Risk dashboards	Limited querying (filter + aggregate only)	**Distributed SQL-based aggregation and analytics**
Settlement & clearing	Not designed for transactional guarantees	**Yes – fully transactional updates, snapshot, failover**

Final Recommendation Summary

Category	Best Choice
Speed for key-value cache	Redis Enterprise

Category	Best Choice
Distributed ACID transactions	GridGain Enterprise
Real-time dashboards + SQL	GridGain Enterprise
ML feature serving	Redis Enterprise
Streaming analytics + compute	GridGain Enterprise
Ease of use / Dev onboarding	Redis Enterprise
Full data platform replacement	GridGain Enterprise

Redis Enterprise –

Not Suitable Use Cases Compared to GridGain

Use Case / Scenario	Why Redis Falls Short	Why GridGain Excels
1. Distributed ACID Transactions	Redis only supports single-shard transactions with no cross-node ACID compliance.	GridGain supports **distributed ACID** transactions across nodes via two-phase commit. Ideal for banking, trade matching, ledger ops.
2. Complex SQL-Based Analytics	Redis lacks native SQL engine; RedisSearch is limited for joins, group-bys, and relational logic.	GridGain supports **full ANSI SQL**, joins, aggregation, subqueries, and JDBC/ODBC access. Used for VaR, intraday P&L, exposure reporting.
3. Co-located Compute and Data Processing	Redis has no concept of co-located server-side compute; analytics must happen outside.	GridGain enables **in-grid computation** (map-reduce, ML scoring, streaming) **on the same nodes as data** — reducing network latency.
4. Trade Lifecycle Management (IBOR)	Redis can't guarantee atomic multi-key/multi-entity updates across shards.	GridGain enables **transactional ledger updates**, with snapshotting and recovery. Ideal for matching, booking, settlement pipelines.
5. T+0 Settlement & Streaming Event Processing	Redis Streams lacks integration with Spark/Kafka for true streaming ETL.	GridGain includes **native streaming support**, windowed aggregation, and integration with Kafka/Spark for real-time clearing ops.
6. Real-Time Risk Aggregation	No SQL-based aggregations across datasets; slow or manual workaround required.	GridGain's SQL engine + MPP architecture enables **real-time, in-memory VaR, stress tests, and exposure tracking**.

Use Case / Scenario	Why Redis Falls Short	Why GridGain Excels
7. Distributed Ledger with Durability	Redis persistence is snapshot/AOF; requires reloads; risk of partial writes.	GridGain uses **native persistence + journaling + checkpoints** for **instant recovery and durability**, even across thousands of trades.
8. Enterprise Integration (DIH)	Redis requires external ETL for RDBMS/NoSQL integration.	GridGain acts as a **Digital Integration Hub**, ingesting from Oracle, PostgreSQL, Kafka, Hive, etc., and serving data via SQL.
9. Complex Compliance Auditing or Governance	Redis lacks snapshot-level auditability and recovery tooling.	GridGain has built-in **snapshotting, backup, audit logs**, and RBAC integrated into the Control Center.
10. Highly Structured, High-Volume Data Models	Redis performs poorly with complex, relational models (e.g., trades, allocations, counterparties).	GridGain allows **schema-rich models**, joins, and storage of high-volume structured financial data (like nested trade trees or books).

Example: Redis Limitations in Financial Systems

Application	Redis Limitation	Real Consequence
Trade Matching Engine	Cannot guarantee atomic update of order book and ledger across shards	Risk of mismatched trades or partial booking
Intraday VaR Engine	Cannot perform distributed joins or SQL aggregations	Incomplete or delayed risk reporting
Cross-Account Settlement	No multi-entity atomicity or durability guarantees	Funds debited but not credited—violating regulatory rules
Order Audit Trail	No native snapshots or timestamped data history	Difficult to comply with MiFID II, SEC Rule 17a-4
Real-Time Fraud Detection	Features must be pre-joined or precomputed externally	Added latency, inconsistent scoring

When to Use GridGain Over Redis

Choose **GridGain Enterprise instead of Redis Enterprise** when:

- You need **distributed transactions and SQL** on **structured datasets**.
- You're building a **financial ledger**, **risk engine**, or **trading system**.
- Your architecture must support **in-memory persistence**, **streaming analytics**, and **instant recovery**.
- You're integrating **multiple data sources** (e.g. RDBMS + Kafka + APIs).
- You want a **unified platform** for compute + storage + ML pipelines.

Redis Enterprise –

Best Fit Compared to GridGain

Use Case / Scenario	Why Redis Excels	Why GridGain Falls Short
1. Ultra-Low Latency Caching	Redis delivers **sub-millisecond reads/writes** using in-memory storage + optimized modules.	GridGain is fast but adds overhead due to distributed processing layers.
2. Simple Key-Value or Session Storage	Ideal for **session state, JWT tokens, shopping carts, rate limiting**, etc.	GridGain is too heavyweight and complex for these small-scale tasks.
3. Microservice Response Acceleration	Used as a **shared cache** for microservices (e.g. token auth, config lookups).	GridGain introduces latency and ops complexity for simple lookups.
4. Real-Time Leaderboards / Counters	Native support for atomic **INCR, ZADD, ZSCORE, etc.**, perfect for gaming or analytics counters.	GridGain lacks native data types like sorted sets; requires more design effort.
5. Pub/Sub Messaging / Streams	Redis Streams and Pub/Sub support **lightweight messaging** across services with low overhead.	GridGain is designed for streaming + compute, not simple fanout.
6. Real-Time Feature Store for ML	RedisJSON, RedisVector, RedisSearch modules allow fast feature retrieval for model serving.	GridGain can host ML logic but lacks specialized modules for vector search or JSON parsing.
7. Temporary Caching Layer for Expensive Queries	Acts as a **read-through/write-through cache** over RDBMS or APIs.	GridGain can do this but takes more setup, especially for short-lived data.
8. Time-Series Analytics (Basic)	RedisTimeSeries module supports **rollups, downsampling, compact storage**.	GridGain supports streaming but is heavier to manage for basic time-series.
9. API Rate Limiting / Throttling	Native support for token bucket & sliding window logic in-memory.	GridGain would require external logic + cluster overhead.
10. SaaS Tenant Isolation	Namespace isolation via logical databases; can scale per tenant easily.	GridGain is harder to multitenant at small scale.

GridGain Overkill or Not a Fit For:

Application	Why Redis Is Better	GridGain Drawback
Config cache for microservices	Redis has built-in TTLs, instant lookup, and high availability.	GridGain requires cluster coordination, setup for trivial data.
Leaderboards in gaming/ads	Redis Sorted Sets = native fit; optimized for atomic ranking ops.	GridGain lacks native sorted-set functionality.
IoT device telemetry lookup	Redis delivers <1ms lookups for millions of devices.	GridGain is better suited for complex event processing, not key reads.
SaaS user preference storage	Redis works well for per-user metadata, profiles, sessions.	GridGain is excessive for 1–2 KB profile blobs.
Real-time fraud scoring API	RedisFeatureStore serves vector/JSON features in <1ms.	GridGain needs full model/feature ingestion or external pipeline.

When to Use Redis Over GridGain

Choose **Redis Enterprise instead of GridGain Enterprise** when:

- You need **low-latency data access** (sub-millisecond) for high-throughput APIs.
- Your workload is **read-heavy** or focused on **caching, session storage**, or **real-time scoring**.
- You're building **stateless, microservice-based systems**.
- You don't need **SQL**, **distributed transactions**, or **streaming analytics pipelines**.
- You want **easy deployment**, lower ops cost, and rapid development cycles.

Summary: When Redis Is the Better Fit

Redis Enterprise Best For	GridGain Overkill Because…
Microservices, token caches, and auth state	Requires no SQL or ACID
Leaderboards, real-time counters	Native commands like INCR, ZADD, ZREVRANK
Pub/Sub or fanout messaging between components	GridGain's compute grid is too complex for simple messages
Feature stores for real-time inference	RedisVector and JSON modules offer instant lookup

Redis Enterprise Best For	GridGain Overkill Because…
Rate limiting, quotas, API TTL enforcement	Built-in rate-limiting algorithms
Time-series with rollups and retention	Simple and fast with RedisTimeSeries