Análise e Benchmarking de Base de Dados

Resumo

Este relatório analisa e compara as opções de persistência de dados para um stack de IA on-premises, cobrindo várias bases de dados vetoriais e relacionais.

A avaliação centra-se em desempenho, escalabilidade, consumo de recursos, complexidade operacional e integração com frameworks LLM / RAG.

Conclui-se com a recomendação de **Qdrant** para a camada vetorial e **PostgreSQL** para a camada relacional.

17/04/2026

Funds & Investments – Axians DC

axians

EXÉRCITO

VINCI ENERGIES

Índice

1 Introdução 4
2 Análise e Benchmarking 5
2.1 Conceitos e Métricas de Avaliação 5
2.2 Base de Dados Vetorial 5
2.3 Base de Dados Relacional 8
3 Conclusão 10
4 Referências 11

1 Introdução

O seguinte relatório documenta a análise e comparação das opções de base de dados para o stack de IA desenvolvido on-premises. O sistema assenta num servidor com GPU NVIDIA H100, onde se encontram disponíveis outros serviços, incluindo modelos LLM on-prem.

O âmbito deste relatório cobre dois componentes com requisitos distintos. O primeiro é a base de dados vetorial, responsável pelo armazenamento e pesquisa de *embeddings* no contexto de RAG (Retrieval-Augmented Generation). O segundo é a base de dados relacional, responsável pelos dados estruturados da aplicação: conversas, utilizadores, sessões, entre outros. Para cada componente, foram identificadas e analisadas as opções relevantes, tendo em conta os critérios de desempenho, escalabilidade, consumo de recursos, complexidade operacional e integração com os frameworks de desenvolvimento utilizados, em particular o LangChain.

2 Análise e Benchmarking

2.1 Conceitos e Métricas de Avaliação

Para facilitar a análise e comparação que serão apresentadas de seguida, esta seção introduz os conceitos técnicos utilizados ao longo do relatório, como latência, throughput, entre outros.

- **Latência** é o tempo que decorre entre o envio de um pedido e a receção da resposta. É habitualmente medida em milissegundos (ms) e expressa em percentis: p50 representa o tempo mediano (metade dos pedidos é mais rápido), p95 e p99 representam os tempos dos 5% e 1% de pedidos mais lentos, respetivamente. Os valores de cauda (p95, p99) são os mais relevantes para aplicações de utilizador final, pois definem a experiência nos momentos de maior carga.

- **Throughput** mede a capacidade de processamento de um sistema, expressa em pedidos por segundo (RPS) ou queries por segundo (QPS). Enquanto a latência mede a velocidade de resposta a um único pedido, o throughput mede quantos pedidos o sistema consegue processar em simultâneo. Um sistema pode ter baixa latência em pedidos isolados, mas throughput limitado sob alta concorrência.

- **Concorrência** refere-se ao número de pedidos ou processos a executar em simultâneo. Baixa concorrência corresponde a poucos utilizadores ou processos a aceder ao sistema ao mesmo tempo; alta concorrência corresponde a muitos acessos simultâneos. Os sistemas de base de dados têm comportamentos de desempenho distintos conforme o nível de concorrência, sendo comum que a latência aumente e o throughput sature a partir de um determinado ponto.

- **Pesquisa híbrida** (utilizada na base de dados vetorial) combina dois métodos complementares de pesquisa vetorial: pesquisa densa (dense), baseada em vectores numéricos de alta dimensão que capturam semântica, e pesquisa sparse, baseada em termos exatos como BM25. A combinação melhora a relevância dos resultados, especialmente em corpus com linguagem técnica ou terminologia específica, onde a pesquisa puramente semântica pode não ser suficiente.

- **Escalabilidade vertical** consiste em aumentar os recursos de um único servidor (mais RAM, mais CPU, discos mais rápidos) para acomodar crescimento de carga ou volume de dados. É a abordagem mais simples operacionalmente, mas tem limites físicos e de custo. **Escalabilidade horizontal** consiste em distribuir a carga por múltiplos servidores ou nós, o que elimina os limites de um único servidor, mas introduz complexidade de coordenação e operação. Ambos pontos são importantes para entender de que forma será possível crescer a infraestrutura.

2.2 Base de Dados Vetorial

As bases de dados vetoriais são componentes especializados no armazenamento e pesquisa de vetores de alta dimensão, produzidos pelos modelos de *embeddings*. Num sistema RAG, a qualidade e a velocidade desta pesquisa têm impacto direto na relevância das respostas geradas pelo modelo de linguagem. A avaliação das opções centra-se em seis critérios: latência de pesquisa, suporte à pesquisa híbrida, escalabilidade, consumo de recursos, complexidade operacional e integração com frameworks LLM e RAG.

Foram consideradas as seguintes opções open-source para deployment on-premises: Qdrant, Weaviate, PGVector (extensão do PostgreSQL) e Milvus. Foram excluídas do âmbito da análise as opções sem suporte on-premises ou sem licença open-source, como o Pinecone, o Oracle 23ai e o S3 Vector Bucket. O Elasticsearch, embora tecnicamente viável, foi excluído por ser sobredimensionado para o caso de uso em questão. O MongoDB foi excluído por razões de licenciamento: a versão Community Server, utilizada em contextos on-premises, é distribuída sob a licença SSPL, que não é reconhecida como open-source pela Open Source Initiative, não cumprindo assim os requisitos do projeto.

2.2.1 Qdrant

O Qdrant é uma base de dados vetorial desenvolvida em Rust, com foco em desempenho e eficiência de recursos. Utiliza o algoritmo HNSW (Hierarchical Navigable Small World) como estrutura de índice principal, com uma implementação própria denominada Filterable HNSW, que integra a filtragem por metadados diretamente no grafo de pesquisa, evitando a necessidade de pré ou pós-filtragem e preservando a precisão, mesmo com filtros restritivos.

Os benchmarks publicados pelo Qdrant [1] demonstram que o Qdrant atinge as latências mais baixas e o maior throughput na generalidade dos cenários testados, incluindo datasets com 1 milhão de vectores de 96 e 768 dimensões. Latências na ordem de 1 ms (p50) e 4 ms (p95) são consideradas excelentes para aplicações RAG em tempo real, onde respostas abaixo dos 100 ms são percebidas como imediatas pelo utilizador.

O suporte à pesquisa híbrida, combinando vetores densos com vetores esparsos via BM25, está disponível de forma nativa com as frameworks consideradas em Python (em específico, Langchain). O deployment on-premises resume-se a um único contentor Docker, sem dependências externas.

Concluindo, o Qdrant é a opção com melhor equilíbrio entre desempenho, simplicidade operacional e capacidade de escalar horizontalmente via sharding nativo, sendo a escolha mais adequada para o perfil deste projeto.

2.2.2 Weviate

O Weaviate é uma base de dados vetorial open-source desenvolvida em Go, com suporte à pesquisa híbrida, multi-tenancy nativo e uma interface GraphQL para consultas estruturadas. Utiliza igualmente o algoritmo HNSW como estrutura de índice principal. Do ponto de vista funcional, é a opção mais próxima do Qdrant em termos de capacidades.

A principal distinção operacional face ao Qdrant reside no modelo de dados "schema-first": cada coleção no Weaviate requer a definição prévia das propriedades e dos seus tipos de dados. Esta abordagem oferece maior controlo sobre a estrutura dos dados, mas introduz uma sobrecarga de gestão quando os tipos de documentos ingeridos são de natureza variada ou evoluem frequentemente.

O Weaviate mantém o índice HNSW inteiramente em memória [2], o que poderá resultar num consumo de RAM desproporcionalmente elevado face ao volume de dados armazenados. Existem mecanismos de compressão (Product Quantization, Binary Quantization) que reduzem o consumo, mas com impacto na precisão da pesquisa.

Concluindo, o Weaviate apresenta capacidades técnicas equivalentes ao Qdrant em vários critérios, mas o consumo de memória elevado e o modelo "schema-first" tornam-no menos adequado para o perfil

deste projeto, onde a simplicidade operacional e a eficiência de recursos são prioritárias.

2.2.3 PGVector

O PGVector e o pgvectorscale são extensões do PostgreSQL, não bases de dados independentes. O PGVector adiciona ao PostgreSQL o tipo de dados vetor e suporte à pesquisa por similaridade ANN via índice HNSW. O pgvectorscale, desenvolvido pela Timescale, complementa o PGVector com o algoritmo StreamingDiskANN, que armazena o índice em disco em vez de memória, permitindo escalar para volumes de vetores que, de outra forma, exigiriam quantidades de RAM. Ambas as extensões funcionam dentro do processo PostgreSQL, sem serviços adicionais, o que é a sua principal vantagem operacional face às soluções dedicadas.

Um benchmark comparativo [3] com 50 milhões de vetores de 768 dimensões demonstrou que o Qdrant apresenta melhor latência de cauda face ao PGVector, contudo ambos os valores estão abaixo dos 100 ms, o que é considerado adequado para aplicações RAG. Em contrapartida, o PGVector atinge um throughput 11 vezes superior em alta concorrência (471 QPS vs 41 QPS), o que é uma vantagem significativa em cenários com muitos utilizadores simultâneos. Relativamente à construção do índice na base de dados, o tempo com pgvectorscale foi cerca de 11,1 horas, e 3,3 horas com Qdrant, o que será relevante em cenários de ingestão frequente.

O PGVector é a opção de menor risco operacional e a mais simples de adotar se a infraestrutura já incluir PostgreSQL. Para o perfil deste projeto, o PGVector é uma opção válida para um conjunto de documentos de dimensão moderada, mas apresenta limitações de escalabilidade horizontal que o tornam menos adequado para crescimento rápido do corpus documental.

2.2.4 Milvus

O Milvus é uma base de dados vetorial distribuída, concebida para escalabilidade massiva. Suporta múltiplos tipos de índice, incluindo HNSW e IVF (Inverted File Index), pesquisa híbrida e multi-tenancy. É a opção mais escalável, adequada para lidar com documentos na ordem dos mil milhões de vetores.

O Milvus destaca-se no tempo de indexação, sendo a opção mais rápida neste critério. No entanto, os benchmarks do Qdrant demonstram que o Milvus não mantém o mesmo nível de desempenho em RPS e latência quando os *embeddings* têm dimensões elevadas ou quando o número de vetores aumenta significativamente.

A principal limitação do Milvus para o perfil deste projeto é a complexidade operacional. O deployment requer etcd para coordenação de metadados e MinIO para armazenamento de objetos, implicando a gestão de pelo menos três serviços adicionais [4]. Os requisitos de hardware recomendados para produção são de, pelo menos, 8 núcleos de CPU e 32 GB de RAM, com 64 GB recomendados para datasets com mais de 10 milhões de vetores [4]. Em contextos de produção reportados pela comunidade, instâncias standalone com carga moderada chegaram a consumir a totalidade dos recursos de servidores com 32 vCPUs e 128 GB de RAM após um simples restart, obrigando a sucessivos aumentos de hardware.

O Milvus é a escolha correcta para projetos com corpus de grande escala e com equipas com capacidade para operar sistemas distribuídos. Neste caso, a sobrecarga operacional é elevada e os benefícios de escalabilidade não justificam o custo de complexidade.

2.2.5 Comparação e Conclusão

As cinco opções analisadas diferem significativamente em termos de maturidade, complexidade operacional e adequação ao perfil deste projeto.

O Milvus é a opção mais escalável do conjunto, mas a complexidade operacional é excessiva para o perfil atual do projeto, visto que requer a gestão de pelo menos três serviços adicionais e hardware significativo.

O PGVector com pgvectorscale é a alternativa mais interessante para fases iniciais: elimina a necessidade de um serviço extra ao reutilizar o PostgreSQL existente, e apresenta latências sub-100 ms adequadas para RAG. A principal limitação é a escalabilidade horizontal, que não existe de forma nativa, tornando-o menos adequado para crescimento rápido do corpus documental.

O Qdrant destaca-se como a opção com melhor equilíbrio global. Apresenta as latências mais baixas na generalidade dos cenários testados, suporte nativo à pesquisa híbrida, escalabilidade horizontal via sharding nativo e deployment simplificado num único contentor Docker sem dependências externas. O consumo de recursos é reduzido graças à implementação em Rust, o que é relevante num servidor onde a GPU e parte da RAM estão reservadas ao vLLM.

O Weaviate é funcionalmente equivalente ao Qdrant em vários critérios, mas os problemas reportados de consumo de memória elevado por manter o índice HNSW inteiramente em RAM, representam um risco operacional num servidor com recursos partilhados.

Concluindo, o **Qdrant** é a escolha recomendada para este projeto.

2.3 Base de Dados Relacional

A base de dados relacional é responsável pelo armazenamento dos dados estruturados da aplicação: histórico de conversas, perfis de utilizadores, sessões, configurações, entre outros.

Das opções consideradas, o MongoDB foi excluído da análise, devido ao licenciamento, e o SQLite foi excluído por razões operacionais (não suporta escritas concorrentes de múltiplos processos de forma segura, sendo explicitamente desaconselhado em qualquer deployment de produção). A análise foca-se assim no PostgreSQL e MariaDB.

O PostgreSQL é o sistema de gestão de bases de dados relacionais open-source mais adotado em contextos de produção, com alta maturidade. Esta base de dados apresenta desempenho robusto em workloads OLTP (online transacional processing) mistos [5], adequado para aplicações de chatbot com centenas de utilizadores simultâneos. Além disso, o PostgreSQL suporta o tipo de dados JSONB com indexação eficiente, permitindo armazenar estruturas semi-estruturadas sem migrações frequentes de schema.

O MariaDB é igualmente uma solução madura e open-source, com particular aptidão para operações de leitura e escrita simples de alta velocidade e menor complexidade operacional em cenários de baixa carga. Contudo, em benchmarks independentes, valida-se que o PostgreSQL supera o MariaDB em 55% em workloads OLTP, inserindo 84% mais ordens com apenas 39% mais transações [6]. O MariaDB apresenta maior consistência de throughput, mas a um nível inferior.

Concluindo, o PostgreSQL e o MariaDB são ambas soluções robustas e operacionalmente simples em contextos on-premises. A distinção determinante para este projeto reside no tipo de workload: o MariaDB

destaca-se em operações de leitura e escrita simples e de alta velocidade, enquanto o PostgreSQL apresenta melhor desempenho em workloads mistos com queries complexas e acessos concorrentes intensivos. Para o perfil de dados e utilização deste projeto, onde coexistem estruturas relacionais e semi-estruturadas com acesso concorrente de múltiplos utilizadores, o **PostgreSQL** é a escolha mais adequada.

3 Conclusão

Neste relatório, foi feita a análise e comparação das principais opções disponíveis para a camada de persistência de dados, cobrindo seis bases de dados vetoriais e duas bases de dados relacionais, suportada por dados de benchmarking mencionados.

Para a base de dados vetorial, o **Qdrant** destacou-se como a opção com melhor equilíbrio entre latência, suporte nativo à pesquisa híbrida, consumo de recursos, simplicidade operacional e capacidade de escalabilidade horizontal. O PGVector é uma alternativa válida para fases iniciais com corpus reduzido. O Weaviate foi excluído pelo consumo de memória elevado, especialemente no servidor com recursos partilhados com o vLLM, e o Milvus foi excluído pela complexidade operacional excessiva para o perfil atual do projeto.

Para a base de dados relacional, o **PostgreSQL** é a escolha clara, com o melhor desempenho sob carga concorrente e escalabilidade progressiva bem estabelecida

A combinação recomendada é Qdrant para a camada vetorial e PostgreSQL para a camada relacional. Este stack é inteiramente on-premises, open-source, operável num único servidor via Docker Compose, e pode escalar componente a componente à medida que os requisitos evoluem.

4 Referências

[1] Qdrant. Vector Search Benchmarks. https://qdrant.tech/benchmarks, 2024.

[2] Weaviate. Weaviate Documentation: Resources. https://docs.weaviate.io/weaviate/concepts/resources.

[3] Timescale. Pgvector vs. Qdrant - 50M Embedding Benchmark. https://tigerdata.com/blog/pgvector-vs-qdrant, May 2025.

[4] Milvus. Milvus Documentation: Prerequisites Docker. https://milvus.io/docs/prerequisite-docker.md.

[5] Amazon Web Services. The Difference Between MariaDB and PostgreSQL. https://aws.amazon.com/compare/the-difference-between-mariadb-and-postgresql.

[6] Data System Reviews. PostgreSQL vs MariaDB Performance Comparison. https://datasystemreviews.com/postgresql-vs-mariadb-performance-comparison.html, 2024.