Vespa.ai

We Make AI Work

Scaling Smarter: Vespa's Approach to High-Performance Data Management

Vespa.ai

We are a platform for building and running real-time AI-driven applications for search, recommendation, personalization, and RAG. It enables enterprise-wide AI deployment by efficiently managing data, inference, and logic, handling large

Contents

Document
Scaling Smarter: Vespa's Approach to High-Performance Data Management

2025

↗ All contents contain linked titles.

Scaling Smarter: Vespa's Approach to High-Performance Data Management

Introduction

Processing data at any scale with low latency is a challenging problem that requires an unusual systems architecture. Traditional systems rely on moving data to a compute node or cluster when the data is needed, but this does not work when a lot of data is needed quickly because the amount of data that can be moved over the network in time is so limited. While network capacity has improved over time, processing capacity is increasing faster, so this problem has become worse over time.

Businesses today have use cases that require low latency in combination with seamless scaling, zero downtime, and cost optimization without overburdening engineers. This ebook explores how Vespa's advanced architecture optimizes data storage, processing, and performance at scale.

Vespa: Real-Time Data Processing for Scalable Applications

Vespa is a powerful real-time data processing platform designed for high-performance search, recommendation, and other low latency data-driven applications. It supports real-time data ingestion and querying, integrates storage and compute functions, and offers scalability and fault tolerance. Its flexible query and data model enables advanced data processing and delivery of high quality results.

By simplifying complex data management tasks, Vespa allows developers to build scalable applications without manually handling time-consuming and mundane technical details. Applications can be updated and resized without downtime, ensuring continuous availability and improvement. Vespa processes complex queries in real-time, keeping data up-to-date even as it changes rapidly, helping businesses stay competitive while streamlining application development and operations.

Core Concepts of Vespa's Architecture

Data Distribution and Processing

Buckets: Logical units of data distribution. Vespa divides data into buckets, which are dynamically assigned to nodes for optimal storage and performance.

Container Clusters: Handle query, result, and write processing, including any application-specific logic. They manage incoming requests and coordinate query execution across the system.

Content Clusters: Store and manage document data and indexes. Content nodes within these clusters hold the actual data and execute data retrieval, ranking, and inference.

Nodes: Servers that store and process data. Nodes work together within clusters to ensure reliability and scalability in both data size and parallel processing capacity.

Data Management and Fault Tolerance

Consistency Model & Replicas: Ensures data availability and fault tolerance through multiple replicas. Vespa supports eventual consistency while prioritizing high availability and partition tolerance.

Replicas and Redundancy in Data Management: Ensures high availability and fault tolerance by maintaining multiple copies of data across different nodes. If a node fails, Vespa automatically uses a replica to maintain service continuity and re-generate replicas.

Data Modeling and Performance

Document Model: Represents data as structured documents with key-value pairs. Each document follows a defined schema, enabling efficient indexing and retrieval.

Ideal State: A system state where all data is optimally distributed across nodes, ensuring maximum availability and performance. Vespa continuously works toward this state in the face of changes such as loss of nodes or application rescaling.

Read/Write Separation: Optimizes performance by separating indexing (write) and query (read) operations. This separation allows for concurrent data updates and fast search capabilities.

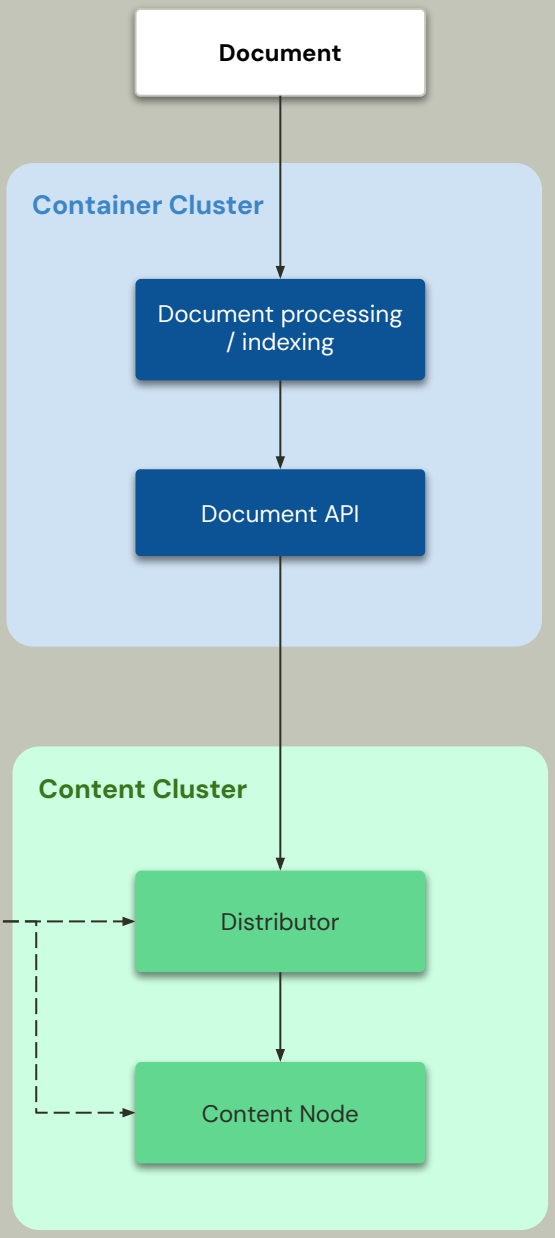
System Configuration

Application Package: A deployable configuration containing services.xml, schemas, ranking profiles, machine-learned models, components, and all other settings that together define a Vespa application.

Schemas: Defines document types, fields, and attributes. It specifies how data is structured, indexed, and ranked within Vespa.

services.xml: Configures services, clusters, and nodes in Vespa. It defines the system's architecture and capabilities.

Processing Incoming Data



The **document model** represents data as structured documents with key-value pairs. Each document follows a defined schema, enabling efficient indexing and retrieval.

Container Clusters handle query, result, and write processing, including any application-specific logic. They manage incoming requests and coordinate query execution across the system.

Content Clusters store and manage document data and indexes. Content nodes within these clusters hold the actual data and execute data retrieval, ranking, and inference.

The **distributor** calculates the correct content node using the distribution algorithm and the cluster state.

The **cluster controller** manages the state of the distributor and content nodes. This cluster state is used by the document processing chains to know which distributor to send documents to, as well as by the distributor to know which content nodes should have which bucket.

Nodes are servers that work together within clusters to ensure reliability and scalability in both data size and parallel processing capacity.

Data Ingestion: Real-Time and Scalable

Vespa enables fast and efficient data ingestion through async streaming APIs and a unique distributed real-time indexing design, where recent changes are indexed in memory and later consolidated on disk. The process includes parsing incoming data into Vespa's document model, processing the document according to what is specified in the application schema, distributing data to content nodes for fault tolerance and scalability, and persisting the data and applying the required index changes on those nodes.

The whole process completes for each document or document update before acknowledging the operation to the client, ensuring that all operations are persisted and observable in real time. Since the process uses streaming and async responses end-to-end (using HTTP2 to the client), there is no need for a separate batch API.

Write operations to particular fields are applied without changing the entire document. This ensures that changes to in-memory metadata fields can be applied cheaply, often reaching a rate of over 100k writes per second. This enables Vespa applications to incorporate fast-changing signals such as remaining advertisement budget or user behavior feedback in real time.

Overall, Vespa's data ingestion process ensures fast, reliable, and scalable data management. Businesses benefit from immediate data availability, operational continuity, and the flexibility to scale effortlessly as their data and traffic increase.

Vespa's Approach to Efficient Data Storage and Management

Vespa distributes data across multiple servers, called nodes, to ensure reliability and scalability. This setup provides fault tolerance, meaning that when a node fails, queries and writes continue to workload and data copies are automatically distributed over the remaining nodes. More nodes can be added as data or traffic grows, and removed when it shrinks, enabling your applications to track demand without either wasting resources or risking running out of capacity.

Storing and Processing Data

Content clusters in Vespa store data, maintain indexes and execute queries, including using machine-learned models for ranking and inference. When a query is executed, the content clusters locate relevant data, apply ranking models, and return organized results in real time.

Organizing Data for Optimal Distribution

Vespa organizes data into virtual buckets, which are assigned to nodes using an ideal state algorithm. This is used to ensure both distributed and consistent management of data, and balanced distribution of data copies. As the system scales, whether by adding or removing/losing nodes, buckets are automatically rebalanced in the background to maintain optimal performance. This process happens without disrupting queries or writes.

Distributors: Managing Data Locations

While nodes store the actual data, distributors manage a real-time map of where buckets are located. They keep this map in memory, tracking details like document counts, storage usage, and metadata. If a distributor restarts, it consults the nodes to rebuild its map, removing the need for a central master server.

When data-related actions like adding or updating, retrieving, or deleting documents occur, Vespa automatically routes requests to the correct nodes based on the bucket's location. This ensures efficient, balanced data management as the system grows or changes. All nodes in a content clusters plays dual roles as both distributors and storage nodes.

Processing Data and Queries

Vespa also uses stateless container clusters to process incoming data, handle search queries, and run application-specific logic. Depending on business needs, developers can run all tasks within a single container cluster or split them into specialized clusters for data processing, query execution, and custom logic.

After processing a query or data operation, the container cluster routes it to the appropriate content nodes that

store the relevant data. If external services are required, Vespa seamlessly integrates additional components to retrieve and combine the necessary data for a comprehensive response.

Handling Growing and Changing Data Seamlessly

As business needs evolve, Vespa simplifies data management by automating application updates, scaling, and consistency management. Developers can quickly deploy new versions of their applications by updating the application package. Vespa automatically distributes updates to the right servers, restarts services when needed, and keeps the system running smoothly without manual intervention. This makes rolling out new features and improvements fast, hassle-free, and reliable. They can deploy changes quickly, iterate frequently, and maintain optimal performance—all while reducing operational costs.

Effortless Data Growth Management

Vespa's architecture is designed to handle growing and changing data with minimal manual effort:

Automatic Data Redistribution: Vespa uses an intelligent redistribution algorithm similar to CRUSH, ensuring minimal data movement when nodes are added or removed. This minimizes service disruptions while keeping the system balanced and efficient.

Automated Scaling: Vespa can automatically expand capacity and rebalance data without downtime as data volumes or load increase. Engineers no longer need to spend weeks planning capacity upgrades or service expansions.

Transparent Scalability: Scaling does not require service interruptions. Vespa automatically handles node failures, resource scaling, and adaptive storage expansion behind the scenes.

Vespa's Consistency Model

Writes in Vespa are immediately observable in all subsequent operations, making it fully consistent during normal operations. In failure situations Vespa will prioritize availability over consistency when there is a conflict, ensuring that all requests receive a response even if some observed data may not be consistent. Formally, Vespa is an AP (Availability and Partition tolerance) system under the CAP theorem.

Overcoming Limitations of Traditional Systems

In systems like Elasticsearch, the number of shards must be decided at index creation, which doesn't easily change over time. These shards are then distributed across your cluster nodes. If the number of shards doesn't align well with the number of nodes, you can end up with hotspots—nodes that are overloaded while others remain underutilized.

Key Challenges

As your cluster grows and scales, this static sharding model creates five main challenges, each with significant trade-offs:

Limited Scaling Flexibility (Scaling by Divisors): You can only scale effectively by adding nodes in multiples that evenly divide the total number of shards (including replicas). This leads to reduced scalability and flexibility due to rigid shard-to-node ratios.

Oversharding (Excessive Shard Creation): Creating more shards than needed (e.g., 1,000 shards for 100 units of data) to simplify scaling, which leads to higher search overhead and degraded performance. Query performance suffers as each search must check all shards, increasing latency.

Imbalanced Distribution (Add Nodes Without Resharding): When adding nodes to a cluster, if the total number of shards isn't evenly divisible by the new node count, you will encounter hotspots. This challenge is unavoidable in large clusters (50+ nodes), where managing an excessive number of shards or frequently changing divisors isn't practical.

Hotspots may require occasional manual intervention, such as redistributing hot shards when multiple resource-intensive ones are assigned to the same node.

Shard Splitting and Reindexing (Manual Restructuring): Splitting large shards or reindexing requires downtime, extra capacity, and operational expertise. This complex and costly process also involves the risk of inconsistent data flow during reindexing.

Growing Transaction Log (TLOG) Issues: In traditional systems, copying a shard requires storing incoming updates in a transaction log (TLOG). For large shards, this process can be slow, causing the TLOG to grow excessively, which risks delays or failures. As a result, systems with fixed shard sizes often struggle to handle shards larger than 50–200GB, depending on hardware and network speed.

Overcoming the Limitations of Fixed Sharding with Vespa

Overall, fixed sharding fundamentally restricts your ability to scale efficiently and reliably. Each approach to address the imbalance (divisors, oversharding, imbalance tolerance, or splitting/reindexing) introduces operational costs, degraded performance, or limited flexibility.

By contrast, you don't need to worry about managing shards manually in Vespa. When you add or remove nodes, Vespa automatically makes the minimal necessary changes across all nodes to stay well distributed. Since buckets are just a virtual management unit, Vespa can ensure it has a great many more buckets than nodes, which makes it possible to avoid any hotspot issues. This approach is more reliable because it avoids the common problem of handling updates during shard copying, allowing for smoother scaling and more efficient data distribution.

Vespa eliminates these challenges through a dynamic, automated, and scalable architecture. Its dynamic data distribution removes the need for static shard allocation, while built-in automatic data rebalancing ensures smooth scaling with minimal downtime. This approach reduces operational complexity, enabling businesses to scale effortlessly while maintaining high availability and performance—even as data and workloads grow.

Conclusion

Vespa delivers low latency processing of data at any scale by moving computation to data rather than the other way around. Its dynamic data redistribution, fault tolerance, and real-time query processing simplify operations and enable seamless scalability. By integrating storage and compute functions, Vespa reduces operational complexity and cuts infrastructure costs.

For businesses, Vespa empowers organizations to deliver personalized, real-time search results, recommendations, and data-driven experiences. Updating data in real-time keeps customer interactions relevant and engaging, increasing satisfaction and boosting conversion rates. Its automated data management and built-in query processing allow businesses to deploy, scale, and update applications quickly, reducing development time and accelerating innovation. By consolidating storage and compute functions, Vespa can deliver low latency results at any scale, using hardware optimally to minimize costs.

For engineering teams, Vespa simplifies architecture by unifying search, storage, and data processing into a single, scalable platform. This streamlined design reduces system complexity, allowing teams to manage fewer components while ensuring peak performance. Vespa's fault-tolerant design ensures continuous availability by automatically redistributing data during node failures or maintenance, making it ideal for high-traffic environments. Its flexible query and data model also supports advanced data processing, machine-learned ranking, and customizable search features so developers can tailor indexing, queries, and models to fit specific business needs.

Whether your organization requires real-time search, personalized recommendations, or advanced data processing, Vespa's proven and scalable architecture, make it a powerful platform for powering data-intensive real-time applications with efficiency, reliability, and precision.

Vespa.ai is a platform for building and running real-time AI-driven applications for search, recommendation, personalization, and RAG. It enables enterprise-wide AI deployment by efficiently managing data, inference, and logic, handling large data volumes and over 100K queries per second. Vespa supports precise hybrid search across vectors, text, and structured metadata. Available as both a managed service and open source, it's trusted by organizations like Spotify, Vinted, Wix, and Yahoo. The platform offers robust APIs, SDKs for integration, comprehensive monitoring metrics, and customizable features for optimized performance.

Interested to learn more? We have many different resources and information available through our social platforms

GitHub Twitter LinkedIn YouTube