System Design Day9: Search Engine System Design

🔍 Search Engine Design Summary (with Spring Data + Caching Insight)

📌 Goal

Design a **scalable web search engine** (like Google) capable of:

- Accepting user search queries.
- Returning relevant site results with titles, descriptions, and URLs.

---

1. Core Components Overview

- **Client**: User interface to enter queries.
- **API Layer**: Accepts queries, handles pagination, fetches relevant data.
- **Load Balancer**: Distributes traffic to API servers for horizontal scaling.
- **Database**: Stores metadata about web pages.
- **Blob Store**: Stores raw page content (e.g., S3).
- **Crawlers**: Fetch and process internet pages.
- **URL Frontier**: Manages crawling order and politeness.
- **Indexes**:
  - **Text Index**: Word → URLs with frequency.
  - **Global Hash Index**: Detects duplicate pages.
  - **Shard Key Index**: Fast URL lookups.

---

2. API Design

- Accepts search queries and returns:

    - Title
    - Description
    - URL
- Supports pagination (`page number`).
- Scalable via **load balancer** and stateless API servers.
- **Spring Data** can be used at the API layer for efficient querying of the index/database.
    - Facilitates flexible repository-based querying.
    - Could integrate **caching (e.g., Redis)** to store frequently searched terms or results to reduce DB load and latency.

---

3. **Database & Blob Store**

- **Metadata DB** stores:
    - URL
    - Title & Description
    - Content hash (for deduplication)
    - Last updated timestamp
    - Scrape priority
- **Blob Store**:
    - Stores full HTML/content of pages.
    - Reduces load on DB.
    - Referenced by metadata.

---

4. **Scalability via Sharding**

- Metadata DB is **sharded** by URL (high-cardinality key).
- Enables fast lookups and distributed writes.
- **Global Hash Index**:
    - Hash → URL mappings

    - Helps detect duplicates across shards.
- **Text Index**:
    - Word → URLs (with frequency)
    - Sharded by word
    - Used to retrieve top-matching documents for queries.

---

5. Indexing & Ranking

- **Indexing** builds **fast lookup** capability by creating mappings from words to documents (URLs).
- **Ranking algorithms** (e.g., TF-IDF, PageRank, machine-learned rankers) ensure **relevance** in returned results.
- Lookup (index) + Ranking (score) = **Effective Search Experience**

---

6. Crawler Infrastructure

- **Crawlers** fetch content from the internet.
- Extract new URLs from fetched pages (recursive discovery).
- Must respect **robots.txt** files:
    - Use a **robots.txt cache** to avoid repeated fetches.
- Scale demands:
    - 100B pages, updated every 10 days → ~231K concurrent crawls.
    - Requires 10K+ crawler nodes.
    - Distributed geographically to optimize bandwidth and latency.

---

7. URL Frontier: Managing Crawling Order

Handles:

- **Priority**:
    - High-priority sites (e.g., news) crawled more often.
    - Implemented via **multiple priority queues**.

- **Politeness**:
  - Only **one crawler per host** at a time to prevent overload.
  - Achieved via **host-specific queues** and **timestamped heap**.

✅ URL Frontier Design:

- **Priority Queues**:
  - Categorized by freshness needs.
  - Router chooses queues based on weighted probabilities.
- **Host Queues + Heap**:
  - Each host assigned a queue.
  - Heap tracks **next eligible crawl time** per host.
  - Prevents multiple crawlers from hitting same host simultaneously.
- **Router**:
  - Moves URLs from priority queues to host queues as needed.

---

8. System Flow

🔁 Ingestion (Crawling):

1. Crawler fetches URL from URL Frontier.
2. Checks robots.txt (from cache).
3. Downloads page, hashes content.
4. If unique:
   - Store content in blob store.
   - Store metadata in DB.
   - Update global hash/text indexes.
5. Extract new URLs → URL Frontier.

🔍 Query Flow:

1. User sends query to API.
2. API queries **text index** → retrieves matching URLs.

3. Fetch metadata from DB.
4. Optionally fetch content from blob store.
5. **Ranking applied** to sort most relevant results.
6. **Spring Data** + **Caching** (e.g., Redis) can reduce response time for repeated queries.

---

9. **Scalability & Efficiency**

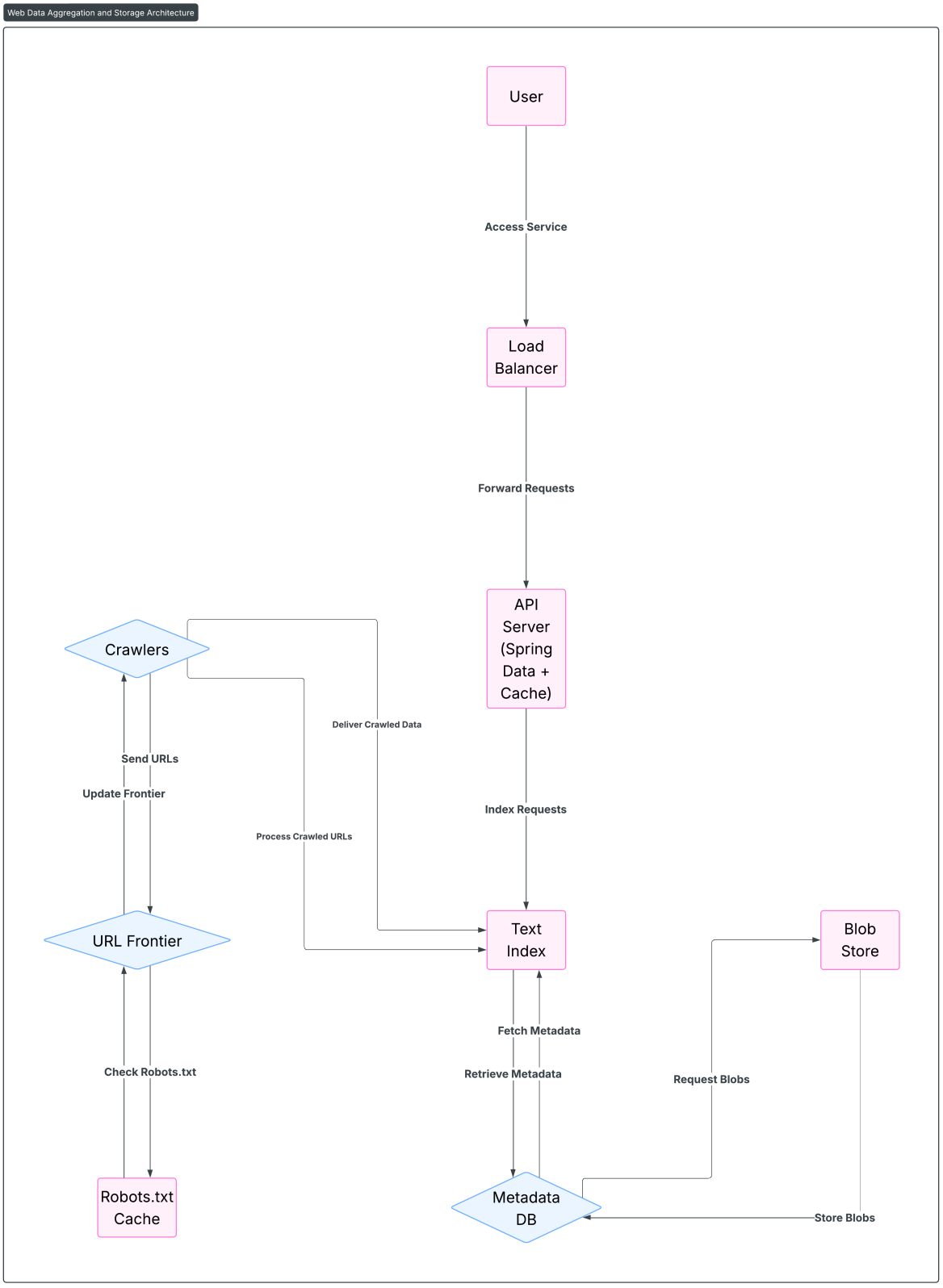
- Metadata DB (~30TB) → Sharded.
- Raw content (~200PB) → Offloaded to blob store.
- Crawler bandwidth: ~2 Tbps → Requires geo-distribution.
- URL Frontier:
  - ~5TB of URLs → Mostly on disk, partial in RAM.
  - Heap in memory is lightweight and recoverable.

---

10. **Possible Extensions**

- **Fault Tolerance**: Recovery if any node (selector, queue, API) fails.
- **Close Duplicate Detection**: Use **shingles** (rather than just hashes).
- **Advanced Indexing Techniques**: TF-IDF, inverted index optimization.
- **Security & Abuse Prevention**: Rate limiting, spam URL filtering.
- **Freshness & TTL Handling**: Ensure search results stay relevant.

✅ Final Design Summary Diagram (High-level Flow)


Web Data Aggregation and Storage Architecture
User
Access Service
Load Balancer
Forward Requests
API Server (Spring Data + Cache)
Crawlers
Deliver Crawled Data
Send URLs
Update Frontier
Process Crawled URLs
Index Requests
URL Frontier
Text Index
Blob Store
Fetch Metadata
Check Robots.txt
Retrieve Metadata
Request Blobs
Robots.txt Cache
Metadata DB
Store Blobs