Distributed Web Crawler

Architecture, Implementation, and Performance Analysis

Saurabh Kumar
2024202029

Nilay Vatsal
2024202003

Y. Kalyan Neeraj
2024202013

International Institute of Information Technology (IIIT Hyderabad)

Introduction & System Goals

A high-performance pipeline designed to discover, fetch, parse, and store web content at scale using a decentralized worker architecture.

Core Technologies

- **Coordination:** Redis (Frontier, Locking)
- **Persistence:** MongoDB (Split Collection)
- **Architecture:** Master-Worker (Autonomous)

Key System Objectives

- **Scalability:** Support 100+ concurrent workers without bottlenecks.
- **Efficiency:** 97% memory savings via Probabilistic Deduplication.
- **Politeness:** Strict adherence to robots.txt and per-domain delays.
- **Fault Tolerance:** No single point of failure; autonomous recovery.

Architectural Patterns

Decentralized Workers

Eliminates master bottleneck. Workers self-coordinate through Redis. Master only handles seeding and monitoring.

Shared-Nothing

Independent HTTP sessions. No inter-worker communication. Crash isolation ensures system stability.

Batch Processing

Buffer-and-flush strategy. Reduces DB round trips. 20x performance improvement over individual inserts.

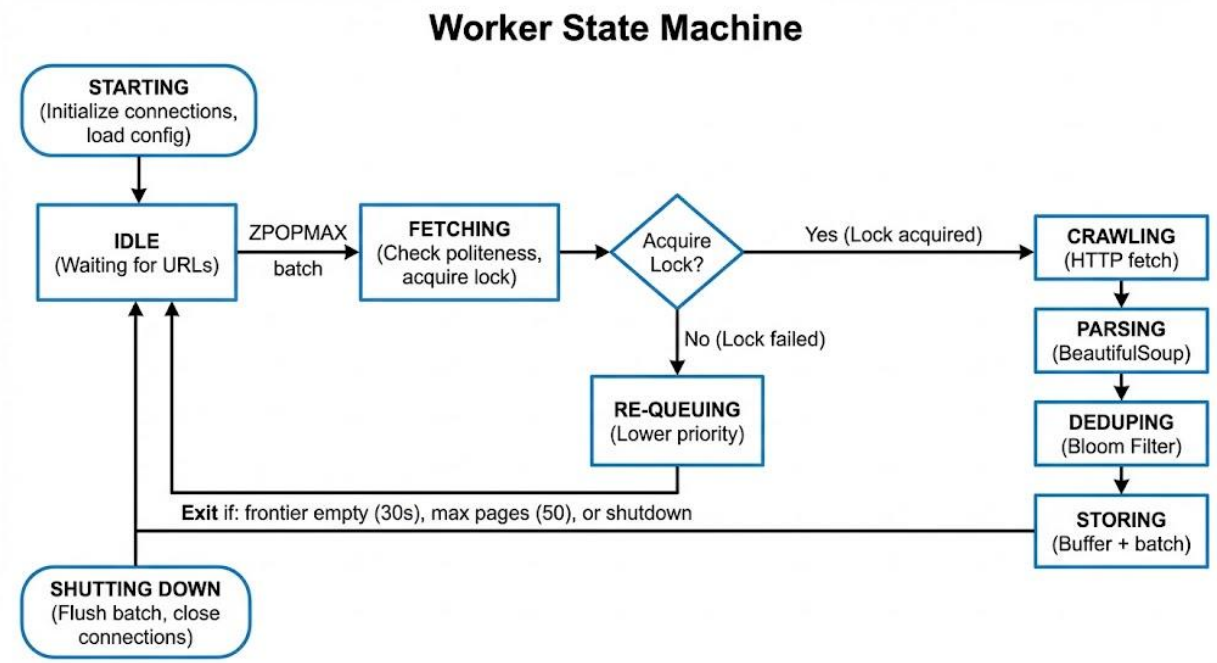
Probabilistic Deduplication (ADR 2)

- **Mechanism:** Bloom Filter via Redis Bitmap.
- **Stats:** 10M URLs tracked in just ~17.2MB.
- **Trade-off:** 0.1% false positive rate (acceptable).
- **Savings:** 97% memory reduction vs. Hash Sets.

High Level Design

Layered Architecture

- **Crawler Layer:** Worker pool executing fetch-parse-dedupe-store pipeline.
- **Coordination Layer (Redis):**
  - Frontier ZSET (Priority Queue)
  - Bloom Filter Bitmap
  - Distributed Locks (SET NX EX)
- **Storage Layer (MongoDB):** Split collections for metadata (fast query) and content (compressed).



Distributed Locking

Distributed Locking Protocol

Ensures only one worker accesses a domain at a time.

Key Pattern

```
lock:{scheme}://{netloc}
```

Acquisition Logic

```
SET lock:domain "1" NX EX {delay}
```

- **NX:** Atomic check-and-set.
- **EX:** Auto-expiry prevents deadlocks.
- **Failure:** If locked, worker re-queues URL with priority penalty (-5).

Priority Scoring Algorithm

Dynamic scoring (0-100) ensures important pages are crawled first while balancing breadth-first exploration.

Depth Penalty

-10 points per level.

Prioritizes shallow, high-value pages over deep links.

Resource Type

-20 points for media/docs.

(.jpg, .pdf, .zip). Focus is on HTML discovery.

Domain Diversity

+5 points bonus.

Applied if domain has <10 pages crawled. Encourages exploration.

Calculation Flow

```
Score = 100 - (Depth*10) - (isMedia*20) - (LongQuery*10) + DiversityBonus
```

Performance: Worker Scaling

Observations

- **Non-Linear Scaling:** Peak throughput achieved at 7 workers (0.60 pages/sec).
- **Diminishing Returns:** Performance degrades beyond 7 workers due to frontier starvation and lock contention.
- **Resource Usage:** CPU remained low (8-19%), confirming the system is Network I/O bound.

Performance: Memory & Storage

Test 2 & 5

Memory Efficiency (50k URLs)

Redis Set	Bloom Filter
3.80 MB	**0.11 MB**

97.1% Memory Savings

Storage Optimization

- **Compression:** zlib level 6.
- **Reduction:** ~50KB Raw HTML → ~10KB Compressed (5-8x reduction).
- **Batching:** 20x Write Speedup (240ms for 50 docs vs 4200ms individually).

Sustained Throughput (90s Test)

- **Stability:** Flat memory usage (~10.5 GB) indicates no leaks.
- **Reliability:** 100% worker uptime during stress test.
- **Predictability:** Throughput variance < 10%.

Challenges & Future Scope

Key Challenges Faced

- **Frontier Starvation:** Limited seed URLs (15) caused idle workers.
  Fix: Increase seed diversity.
- **Domain Lock Contention:** Workers competing for same popular domains.
  Fix: Priority penalization.
- **False Positives:** 0.1% URLs skipped by Bloom Filter.
  Status: Acceptable trade-off.

Future Roadmap

- **HTTP/2 Multiplexing:** Reduce TLS handshake overhead.
- **Redis Cluster:** Shard frontier for high availability.
- **Geo-Distribution:** Deploy workers in multiple AWS regions.
- **Near-Duplicate Detection:** Implement MinHash algorithm.

Q & A

Distributed Web Crawler Project

Scalable

Fault Tolerant

Efficient