Distributed Web Crawler Report

Members:

Samyak Shah(ss4604@rit.edu)
Uzair Islam(ua3415@rit.edu)
Nimisha Mathew(nm9389@rit.edu)

Course: Computer Networks, Fall 2025

1. Introduction

This report presents the implementation of a **Distributed Web Crawler for Website Content Integrity Verification**.

The system automates monitoring of website content by crawling multiple URLs in parallel, computing MD5 hashes of HTML and downloadable content, and reporting results to a central orchestrator. This allows detection of content changes, either due to legitimate updates or unauthorized modifications, without manual intervention.

The project uses a **coordinator–worker architecture** with multiple crawler nodes running concurrently as Docker containers. Workers communicate with the orchestrator via REST APIs, enabling scalability, fault tolerance, and efficient resource utilization.

2. Project Structure

File / Directory	Description
Orchestrator/app.py	Implements the orchestrator API using FastAPI; assigns URLs to workers and collects crawl results
Worker/worker.py	Implements the asynchronous web crawler using aiohttp and asyncio; computes MD5 hashes
docker-compose.yml	Container orchestration for orchestrator, workers, and MongoDB
docs/	Sphinx-generated project documentation
requirements.txt	Python dependencies for both orchestrator and workers

---

3. Environment Setup

- **Python Version:** 3.11
- **Docker Version:** 24+

- **MongoDB Version:** 6.0
- **Required Python Libraries:**
    - fastapi
    - uvicorn
    - pymongo
    - aiohttp
    - beautifulsoup4

Installation:

pip install -r Worker/requirements.txt

pip install -r Orchestrator/requirements.txt

docker-compose up --build

4. System Design

4.1 Architecture

The project follows a **Coordinator–Worker model**:

- **Coordinator (Orchestrator)**
    - Built with FastAPI.
    - Provides endpoints to assign URLs to worker nodes (/get_urls/{worker_id}) and receive results (/post_results/data).
    - Stores crawl results in MongoDB (crawler_db.results).
- **Worker Nodes**
    - Each worker runs as a Docker container.
    - Uses asyncio + aiohttp for concurrent crawling.
    - Computes MD5 hash of HTML pages or PDFs.
    - Sends results back to the orchestrator via REST API.

Workflow:

1. Orchestrator exposes a batch of URLs for each worker.
2. Workers fetch URLs asynchronously, compute MD5 hashes, and send results back.
3. Orchestrator collects results and logs them.
4. MongoDB stores the results for historical comparison.

5. Example Data Flow

Orchestrator Logs:

INFO: Started server process [1]

INFO: Waiting for application startup.

INFO: Application startup complete.

INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)

INFO: 127.0.0.1:51658 - "GET / HTTP/1.1" 200 OK

INFO: 172.18.0.4:57726 - "GET /get_urls/worker1 HTTP/1.1" 200 OK

INFO: 172.18.0.5:39558 - "GET /get_urls/worker2 HTTP/1.1" 200 OK

Received 21 results

INFO: 172.18.0.5:58892 - "POST /post_results/data HTTP/1.1" 200 OK

Received 50 results

INFO: 172.18.0.4:48226 - "POST /post_results/data HTTP/1.1" 200 OK

Worker Logs:

- **Worker 1**

  Crawling https://arxiv.org/search/cs?query=Computer+Network&searchtype=all&abstracts=show&order=-announced_date_first&size=50

  Found 50 PDF links

  Cleanup successful!

- **Worker 2**

  Crawling MIT base page: https://ocw.mit.edu/courses/6-829-computer-networks-fall-2002/pages/lecture-notes/

  Found 21 MIT resource pages

  Found 21 MIT PDF files

  Cleanup successful!

MongoDB Verification:

> use crawler_db

> db.results.find().count()

71

```
> db.results.find().pretty()
{
  _id: ObjectId('...'),
  url: 'https://ocw.mit.edu/.../lecture1.pdf',
  file: 'mit_pdfs/lecture1.pdf',
  md5: 'abc123...',
  status: 'success'
}
```

6. Key Features

Feature	Description
Distributed Crawling	Multiple workers fetch URLs concurrently using asynchronous requests
MD5 Hashing	Each resource is hashed to detect content changes
REST API Communication	Workers send results back to orchestrator via HTTP POST
Dockerized Deployment	Orchestrator and workers run as isolated containers; MongoDB persistence
Scalability	Adding more workers increases crawling capacity without code changes

---

7. Example Usage

Starting the System:

docker-compose up --build

Fetching URLs (Worker):

GET /get_urls/worker1

GET /get_urls/worker2

Posting Results:

POST /post_results/data

Body: [{"url": "...", "file": "...", "md5": "...", "status": "success"}]

MongoDB Verification:

docker exec -it mongo mongo

> use crawler_db

> db.results.find().pretty()

---

8. Sample Output

1. **Worker crawling Arxiv PDFs** – 50 links fetched.
2. **Worker crawling MIT lecture notes** – 21 PDFs and pages.
3. **Orchestrator receiving results** – 71 total results collected.
4. **MongoDB data verification** – results stored in crawler_db.results with MD5 hashes.


```
[
  {
    _id: ObjectId('691b9c26288f140f61822638'),
    url: 'https://ocw.mit.edu/courses/6-829-computer-networks-fall-
2002/8ad6d3727269f49f9f678fd823c8957c_L22srm.pdf',
    file: 'mit_pdfs/8ad6d3727269f49f9f678fd823c8957c_L22srm.pdf',
    md5: '9e0c36a406af5a3fe84ab31182094e61',
    status: 'success'
  },
  {
    _id: ObjectId('691b9c26288f140f61822639'),
    url: 'https://ocw.mit.edu/courses/6-829-computer-networks-fall-
2002/44b429c8e1477cb2f322dc156b5a0f41_t2ns.pdf',
    file: 'mit_pdfs/44b429c8e1477cb2f322dc156b5a0f41_t2ns.pdf',
    md5: '61dda2630c2ae0e5fd1f05af62c17d5e',
    status: 'success'
  },
…….etc
{
    _id: ObjectId('691b9c41288f140f6182265e'),
    url: 'https://arxiv.org/pdf/2511.11032',
    file: 'arxiv_pdfs/2511.11032',
    md5: '1b81dcf22834e5978949ffcbef3da6ed',
    status: 'success'
  },
  {
    _id: ObjectId('691b9c41288f140f6182265f'),
    url: 'https://arxiv.org/pdf/2511.11490',
    file: 'arxiv_pdfs/2511.11490',
```

```
    md5: '8c27cd1dba19ec0dd8c433fcdc7044d7',
    status: 'success'
  }
]
```

9. Conclusion

This project successfully demonstrates a **distributed web crawling framework** with integrity verification.

Key achievements:

- Parallel crawling using Dockerized workers.
- REST API orchestration and inter-container communication.
- Content integrity verification using MD5 hashing.
- Persistent storage in MongoDB for historical comparison.