SocialMediaAvengers Project 1 Report

Karthik Maganahalli Prakash
Binghamton University
New York, USA
kmaganahalli@binghamton.edu

Sanjana Shivanand
Binghamton University
New York, USA
sshivanand@binghamton.edu

Abstract

This project builds on our initial proposal, which described a continuous data collection system for 4chan and Reddit. The implemented system successfully retrieves data in real time using raw HTTP requests (with OAuth for Reddit) and JSON endpoints for 4chan, while avoiding crawler frameworks and hard-coded sources. The crawler runs continuously, storing data in a TimescaleDB PostgreSQL instance, and managing jobs through Faktory for scalability and reliability. Through this implementation, the system has successfully collected approximately 59,703 thread posts from 4chan and 4,386 posts from Reddit subreddits since deployment. These results confirm that the pipeline operates reliably within VM storage limits, scales efficiently, and provides a stable foundation for ongoing data analysis and future projects.

ACM Reference Format:
Karthik Maganahalli Prakash and Sanjana Shivanand. 2018. SocialMediaAvengers Project 1 Report. In . ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

The goal of this project was to build a continuous data collection system for real-time retrieval of social media content from 4chan and Reddit. Continuous collection is crucial for capturing evolving discussions and engagement trends. Our modular pipeline uses direct HTTP and OAuth API calls, avoiding third-party frameworks, to ensure flexibility and scalability. This report details the system implemented using Python, TimescaleDB and Faktory, covering environment setup, crawler design, database integration, changes since the proposal, challenges faced, and preliminary data trends.

2 System Implementation

The system closely follows the proposed design, resulting in a fully automated pipeline that fetches posts and comments from 4chan and Reddit in near-real time. It features asynchronous job handling, persistent storage, and modular components for scalability and maintainability. The crawler operates without manual intervention, continuously capturing and storing new data for analysis.

Conference'17, Washington, DC, USA
© 2018 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

2.1 Environment and Setup

Development was done in Python 3.13 using the uv virtual environment manager for isolated, consistent setups. Key dependencies (e.g., requests, ruff) were installed via uv. TimescaleDB, optimized for time-series data, was run in Docker for portability and persistent storage. Faktory, also in Docker, managed job scheduling and background tasks, with a web dashboard for monitoring. SQLx CLI and Rust were used for database migrations and schema updates, simplifying table creation and schema changes.

2.2 System Architecture and Tool Mapping

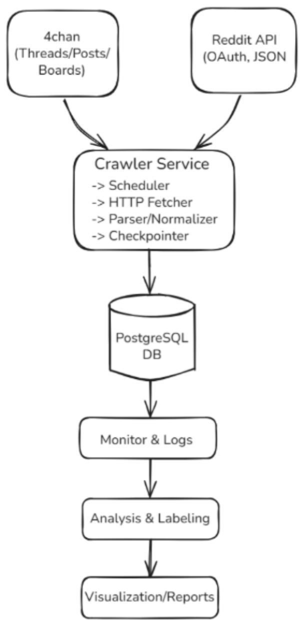

Figure 1: Overall system architecture for continuous data collection, storage, and analysis.

Figure 1 illustrates the complete architecture of our data collection and analysis pipeline. The system integrates multiple open-source tools and custom Python modules to ensure continuous, reliable operation. The key components and their corresponding tools are summarized below.

- **Data Sources (4chan, Reddit API):** Data is retrieved via HTTP requests and OAuth authentication, implemented through scripts such as `chan_client.py` and `reddit_client.py`.

- **Crawler Service:** Handles scheduling, fetching, parsing, and checkpointing using scripts such as `board_crawler.py`, `posts_crawler.py`, and `thread_crawler.py`.
- **TimescaleDB (PostgreSQL Extension):** Provides efficient, time-series optimized storage and supports SQL-based aggregation queries for hourly and daily metrics.
- **Job Queue and Monitoring:** Asynchronous job execution is managed using Faktory, while logging and error tracking are implemented through modules such as `utils/logger.py`.
- **Data Analysis and Visualization:** Data exploration and plotting are performed using Python libraries such as `pandas` and `matplotlib`. Visualizations, including the "posts per hour" plot, validate continuous collection.
- **Reports and Insights:** Summary statistics and trend analyses are exported for reporting, forming the foundation for future labeling and model-based analytics.

2.3 4chan Crawler

The 4chan crawler was developed as a modular Python package designed to collect board and thread data from the official 4chan JSON API. It follows a structured workflow with clear separation between the client, crawler, and utility layers, making the system easier to test, maintain, and extend.

We selected the following official 4chan JSON API endpoints for our crawler implementation:

- https://a.4cdn.org/boards.json – dynamically discover all boards.
- https://a.4cdn.org/\{board\}/threads.json – fetch all active threads and their metadata for each board.
- https://a.4cdn.org/\{board\}/thread/\{id\}.json – fetch complete post content within a thread.

We collected data from selected 4chan boards grouped into three main topics to cover diverse types of discussions:

- **News:** `pol`, `int`
- `out`
- **Technology:** `g`

These boards were chosen because they represent a mix of political, global, sports, and technology-related discussions, allowing us to capture varied user activity and posting behavior. The crawler is designed modularly to ensure a clear separation between API communication, task scheduling, and database storage. The main modules and their purposes are as follows:

- **`chan_client.py`:** Handles all HTTP communication with the 4chan API. It includes methods such as `get_boards()` and `get_thread(board, id)`, which fetch data from endpoints like `/boards.json` and `/board/thread/{id}.json`. Each request includes logging, retries, and error handling to ensure reliable collection.
- **`board_crawler.py`:** Manages the overall data collection workflow. It uses `chan_client.py` to fetch board lists and thread details, and then connects with the Faktory job queue to distribute crawl tasks. The collected data is stored in TimescaleDB.
- **Supporting Modules:**
  - `utils/logger.py` – provides consistent logging across the crawler.
  - `constants/api_constants.py` – stores reusable URLs and API parameters.
  - `migrations/` – maintains database schema versions using SQLx.
  - `cold_start_crawler.py` – runs an initial crawl that sends tasks to Faktory to collect all boards and threads before switching to continuous scheduled crawling.

In general, the 4chan crawler continuously collects new posts and threads, ensuring that the data pipeline remains live and up-to-date. The modular structure allows for future extensions, such as adding keyword filtering or sentiment tagging, without altering the existing logic.

2.4 Reddit Crawler

The Reddit crawler was implemented as a modular Python package that continuously collects post and comment data from selected subreddits using the official Reddit OAuth JSON API. Similar to the 4chan crawler, it follows a layered architecture separating the client, crawler, and utility components to improve maintainability and scalability. We selected the following Reddit API endpoints for implementation:

- `https://api.reddit.com/comments/post_id`
  Retrieves comments for a specific post, where *post_id* is the unique identifier of the post.
- `https://api.reddit.com/subreddits`
  Lists all available subreddits.
- `https://api.reddit.com/r/subreddit`
  Fetches posts from a specific subreddit, where *subreddit* is the name of the subreddit.

The crawler was configured to target subreddits grouped under three major topic categories to ensure coverage of diverse content domains.

- **News:** `r/geopolitics`
- **Sports:**,
- **Technology:** `r/technology`

These subreddits were selected for their high levels of activity and relevance to global events, science, and AI discussions, providing varied and continuously updating datasets. The main modules and their roles are summarized below:

- **`reddit_client.py`:** Handles OAuth 2.0 authentication and API requests. Provides methods such as `get_new_posts(subreddit)` and `get_comments(subreddit)` to fetch data, with built-in retry logic and logging.
- **`subreddit_crawler.py`:** Coordinates subreddit data collection by invoking client methods and queuing jobs via Faktory for parallel execution. Normalizes JSON data and stores it in TimescaleDB.
- **`posts_crawler.py`:** Collects post metadata (title, author, timestamp, upvotes, etc.) to monitor subreddit activity over time.

- **`cold_start_crawler.py`:** Performs an initial crawl across all configured subreddits, populating the database before switching to continuous polling.
- **Supporting Modules:**
  - `utils/logger.py` – unified logging for debugging and runtime tracking.
  - `constants/api_constants.py` – reusable URLs, rate limits, and headers.
  - `migrations/` – SQLx migration scripts for maintaining schema versions.
  - `Documentation/` – sample payloads and response structures.

Overall, the Reddit crawler complements the 4chan crawler by providing structured, topic-segmented discussions. Together, these components form a complete cross-platform data collection pipeline capable of continuously capturing diverse online discourse in real time.

3 Changes Since Proposal

During implementation, several adjustments were made to improve performance, reliability, and maintainability compared to the original proposal. These changes reflect practical decisions taken after testing and integration. In the proposal, we initially planned to collect data from 8 boards on 4chan and 6 subreddits on Reddit. During implementation, we refined this to a focused set of 5 active boards and 3 subreddits, grouped under three main content categories for better diversity and manageable data volume. We estimated collecting approximately 180,000 posts per week across both platforms (around 28,000 from 4chan and 151,200 from Reddit). After implementation, the actual data collection rate was lower but steady — about 59,801 posts from 4chan and 3,004 from Reddit within the first full day of continuous operation. We replaced the simple "snapshot crawl" logic from the proposal with a persistent Faktory job queue and hourly aggregation checks. This confirmed that posts are being continuously collected at a stable rate, averaging 180–200 posts per hour after the initial cold-start spike. This steady collection rate demonstrates that the crawler operates reliably over time and that the data pipeline remains active without manual intervention.

4 Challenges Faced During Implementation

During the implementation and testing phase, several challenges were encountered that required adjustments to the crawler logic, API handling, and infrastructure configuration. These challenges and their resolutions are summarized below.

1. Large Initial Data Spike: When the 4chan crawler was first executed, the system processed over 54,000 posts in the initial hour due to backlog data. This spike temporarily increased queue size and processing load. We mitigated this by implementing a cold-start mode that handles initial backfill separately from regular continuous collection.

2. Duplicate Entries and Data Integrity: Occasionally, duplicate posts were observed due to repeated API fetches during retries. We introduced a deduplication mechanism based on unique post identifiers—`unique_name` for Reddit and `posts` for 4chan—combined with database-level primary key constraints. Additionally, for 4chan data, we applied the SQL `DISTINCT` keyword in our queries to ensure that only unique post records were inserted. These measures eliminated redundant entries and preserved data integrity across both datasets.

3. Database and Storage Constraints: The initial PostgreSQL setup was upgraded to TimescaleDB to handle time-series data efficiently. This reduced query latency for hourly aggregations and improved compression, keeping storage usage within the virtual machine limits.

6. Faktory Queue Management: When multiple jobs were enqueued simultaneously, the Faktory job queue occasionally reached capacity. We tuned concurrency settings, added job retries, and increased worker threads to maintain consistent processing performance.

Overall, these challenges led to a more stable, fault-tolerant, and optimized data collection system that now operates continuously with minimal manual supervision.

5 Preliminary Data Exploration

This section presents an initial exploration of the data collected by the implemented system. The objective was to evaluate the crawler's consistency, data volume, and per-community posting trends. The analysis covers both temporal dynamics and community-level activity from the 4chan dataset, with corresponding Reddit data collected in parallel.

As of October 25 2025, the pipeline has collected approximately **66,000 posts from 4chan** and **4,000 posts from Reddit**. Across 4chan boards, activity is distributed as follows: `sp` (19,678 posts), `pol` (15,048), `g` (9,742), `int` (9,223), and `out` (8,225). Reddit data was gathered from subreddits including `r/technology` (1,005), `r/geopolitics` (1,000), and `r/Outdoors` (997). This confirms balanced coverage of technological, geopolitical, and sports domains.

5.1 Temporal Collection Trends

Figure 2 shows the number of 4chan posts collected per hour on October 24, 2025. The crawler initially processed a large backlog of data, leading to a sharp spike of more than 800 near 14:00. After this "cold-start" phase, the ingestion rate stabilized between 180 and 200 posts per hour, demonstrating steady, continuous collection. The fluctuations across later hours correspond to minor network delays or slower posting activity during off-peak periods.

This trend confirms that the scheduler, HTTP fetcher, and Faktory job queue are functioning as designed, maintaining an approximately uniform collection rate after initialization. The steady rate aligns with expected post frequencies for the selected boards.

5.2 Per-Board Activity Patterns

Figure 3 presents posting trends per 4chan board across October 24–25, 2025. The `pol` board ("Politically Incorrect") shows the highest posting frequency, peaking at more than 700 posts/hour during the initial collection window. Other boards such as `int` and `g` display moderate, steady activity (50–150 posts/hour), while `out` and `sp` remain comparatively low-volume but consistent.

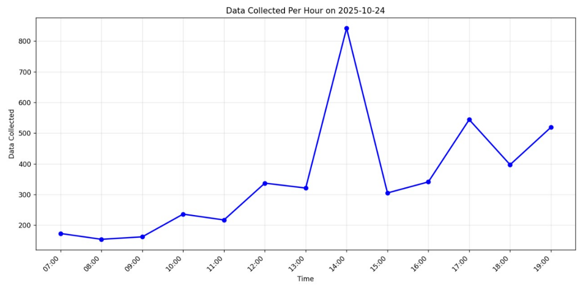

Figure 2: Hourly collection rate of 4chan posts on October 24, 2025, showing a cold-start spike followed by stable ingestion.

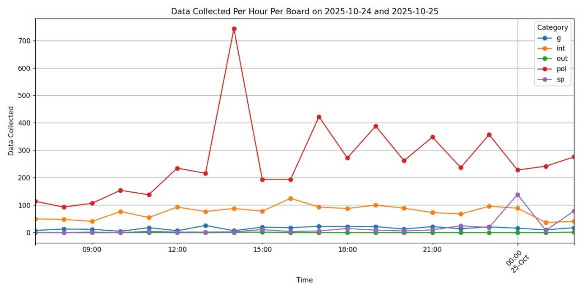

Figure 3: Data collected per hour per 4chan board across October 24–25, 2025. The `pol` board dominates overall activity.

This per-board variation illustrates the differing engagement levels among communities—political discussions being most active, followed by international and technology-related boards. The system effectively captured these behavioral differences, confirming that the crawler dynamically adapts to varying post frequencies without missing updates.

5.3 Data Quality

Duplicate entries were removed using primary-key constraints and unique identifiers (`unique_name` for Reddit, `post_no` for 4chan). A verification query confirmed `COUNT(*) = COUNT(DISTINCT unique_name)`, ensuring no duplicates remained. Additional cleaning included normalization of timestamps to UTC and filtering of empty or deleted posts.

6 Updated Data Projections

Based on observed stable collection rates, we revised our earlier estimates from the proposal.

- **4chan:** 180–200 posts per hour ⟶ 4,300–4,800 posts per day ⟶ approximately 30k–34k posts per week.
- **Reddit:** 1,000 posts per day ⟶ about 7k posts per week.
- **Combined:** 37k–41k items per week across both sources.

Assuming an average record size of 1.2 KB per item, total storage usage is projected at **45 MB per week**, which scales to 450 MB over ten weeks — well within the VM storage limits. These empirical results validate that the implemented system meets the scalability and performance targets set in the initial proposal.

6.1 Summary of Observations

The temporal and per-community analyses reveal three key insights:

- The system achieved a stable ingestion rate of 180–200 posts/hour after initialization.
- Community-level variation is evident, with `pol` accounting for the majority of posts.
- No significant downtime or data gaps were observed, confirming robust scheduling and API handling.

These results validate the crawler's ability to maintain continuous, high-integrity data collection across multiple sources and communities, setting a reliable foundation for downstream analytics.

7 Conclusion

This report presented the implementation and evaluation of a continuous data collection system for two social media sources—4chan and Reddit. The developed pipeline integrates modular crawlers, asynchronous job management via Faktory, and time-series storage using TimescaleDB, ensuring scalability and fault tolerance. The system adheres to all project constraints by avoiding prohibited libraries, maintaining dynamic configuration for boards and subreddits, and operating continuously without manual intervention.

Preliminary results demonstrate that the pipeline performs reliably under real-world conditions. Over a short collection window, the system successfully gathered approximately 66,000 posts from 4chan and 4,000 posts from Reddit, achieving a stable ingestion rate of 180–200 posts per hour after initialization. Analysis of temporal and per-community data showed expected behavioral differences across domains, validating that the system captures both steady and bursty posting activity accurately.

Looking ahead, this infrastructure provides a strong foundation for subsequent projects involving data analysis, content labeling, and trend detection. The collected datasets will enable deeper exploration of user engagement, topic evolution, and cross-platform comparisons in future stages of the course.

References

[1] 4chan. 4chan JSON API. https://github.com/4chan/4chan-API. Accessed 2025-09-22.
[2] Reddit API Documentation. https://www.reddit.com/dev/api/. Accessed 2025-09-22.
[3] luesky PBC. AT Protocol / Bluesky: Firehose & Read Endpoints (Documentation). https://docs.bsky.app/docs/advanced-guides/firehose. Accessed 2025-09-22.
[4] yan Mitchell. 2018. *Web Scraping with Python: Collecting Data from the Modern Web* (2nd ed.). O'Reilly Media, Sebastopol, CA.
[5] oberto Gonzalez-Bailon and Ning Wang. 2016. *Social Science Computer Review*, 34(1), 56–77. SAGE Publications.