

A Hierarchical Orchestration Framework for Parallelized Multi-Entity Web Ingestion and NoSQL Synchronization

Taufique Khan Asadulla Khan^{1,*}, Ankush D. Bambal¹, Kartik S. Kharabe¹, Sonali Nimbhorkar²

Abstract

In the current EdTech sector, gathering structured data from various fragmented web portals is a major challenge for building reliable, real-time data systems. Standard sequential ETL (Extract, Transform, Load) pipelines often encounter significant bottlenecks, as a single failure in one module can halt the entire ingestion process. Furthermore, linear execution is often too slow to meet the demands for large, multi-category datasets. This paper presents a Modular Hierarchical Orchestration framework designed to solve these issues through high-speed, parallel data collection. The system utilizes a centralized “Master–Orchestrator” to manage multiple independent sub-scripts simultaneously each specializing in different domains such as college profile, academic course details, competitive exams, and career trajectories. By running these tasks in parallel, the framework optimizes server CPU utilization and drastically reduces the total ingestion window. The collected data is synchronized and stored in MongoDB NoSQL database, which provides the necessary schema flexibility to handle heterogenous data formats without sacrificing relational integrity. This system was rigorously tested in a real-world industrial environment, where it demonstrated a robust reduction in processing time and superior fault tolerance compared to traditional, single-track scraping methods.

Keywords: Hierarchical orchestration, parallel ETL, Web ingestion, NoSQL, MongoDB, data engineering, scalability, master–slave architecture, fault tolerance, edtech data systems

INTRODUCTION

In the rapidly evolving landscape of Educational Technology (EdTech), the ability to aggregate and synchronize large-scale institutional data is a critical requirement for data-driven decision-making. With thousands of academic institutions, competitive examinations, and career paths distributed across fragmented web portals, traditional data ingestion methods often reach their operational limits [1]. Most conventional Extraction, Transformation, and Load (ETL) pipelines rely on sequential processing, where tasks are executed in a linear queue. However, this approach presents two major drawbacks: high

latency and lack of fault tolerance. A single failure in a sequential script, often caused by a minor change in a website’s Document Object Model (DOM), can halt the entire pipeline, leading to significant delays in data updates and potential loss of intelligence [2].

The complexity is further compounded by the heterogeneous nature of the web data. Each educational portal employs different architectural patterns, ranging from static HTML to dynamic, JavaScript-heavy interfaces. Managing these variations within a single monolithic codebase leads to a “spaghetti code” that is difficult to maintain and nearly impossible to scale. As the demand for real-

*Author For Correspondence

Taufique Khan Asadulla Khan
E-mail: taufiquekhan8833@gmail.com

¹Student, Department. of Computer Applications, GH Raisonni University, Amravati, Maharashtra, India.

²Professor, Department. of Computer Applications, GH Raisonni University Amravati, Maharashtra, India.

Received Date: July 10, 2026

Accepted Date: July 18, 2026

Published Date: August 05, 2026

Citation: Taufique Khan Asadulla Khan, Ankush D. Bambal, Kartik S. Kharabe, Sonali Nimbhorkar. A Hierarchical Orchestration Framework for Parallelized Multi-Entity Web Ingestion and NoSQL Synchronization. Recent Trends in Parallel Computing. 2026; 13(2): 1–6p.

time information increases, particularly regarding rapidly changing exam dates and fee structures, the need for a decoupled, highly available ingestion system becomes paramount [3].

This Paper introduces a resilient, high-speed solution through a Modular Hierarchical Orchestration framework. Implemented within a live industrial environment, the proposed architecture shifts from monolithic scraping models to a modular, parallelized system. As illustrated in Figure 1, the system utilizes a centralized “Master–Orchestrator” design to manage the concurrent execution of multiple independent subextractors. They allow for the simultaneous ingestion of diverse data entities, such as college profiles, course fees, and exam schedules, thereby maximizing server resource utilization and drastically reducing total processing windows [4].

A core component of this framework is the integration of MongoDB, a NoSQL document store. Unlike traditional relational databases, MongoDB’s schema-less architecture provides the flexibility required to handle the often-unstructured nature of educational data. This ensures that variations in the data format between different web sources do not compromise the integrity of the master dataset [5]. This research contributes to the field by demonstrating a scalable alternative to traditional ETL processes and providing a blueprint for industrial grade data acquisition in complex web environments.

LITERATURE REVIEW

The evolution of automated web data extraction has seen a significant shift from basic retrieval to complex distributed orchestration. The following points summarize the current state of research and address the existing gaps in large-scale data ingestion:

Limitations of Sequential ETL

Pipelines

Traditional Extract, Transform, and Load (ETL) systems in the EdTech domain have historically relied on sequential execution [6]. Research indicates that linear processing creates a single point of failure; if a script encounters an unexpected Document Object Model (DOM) change on one website, the entire pipeline stalls. This “brittleness” results in a high maintenance overhead and significant data latency [7].

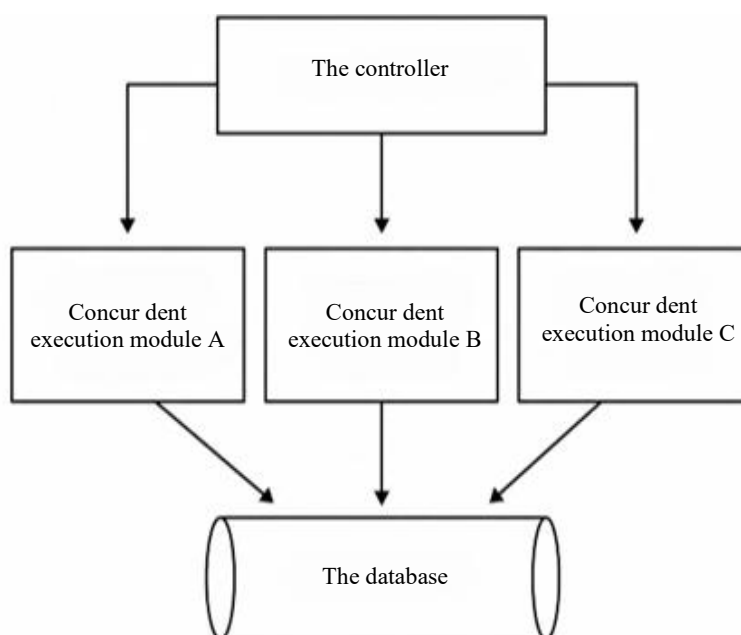


Figure 1. Architectural workflow of the hierarchical orchestration framework, demonstrating the decoupling of the master controller from domain-specific extraction modules and subsequent synchronization to centralized nosql database.

Transition to Headless Browser Automation

Early scraping techniques using libraries such as BeautifulSoup are often insufficient for modern single-page applications (SPAs) built with React or Angular. The industry has shifted toward headless browser automation using tools such as Playwright and Puppeteer. While these tools provide better DOM interaction, managing multiple instances of headless browsers requires significant CPU and memory optimization, a challenge rarely addressed in standard sequential scripts [8].

The Shift from Monolithic and Modular Architectures

Scholars in the field of software engineering have argued that monolithic scraping scripts are difficult to scale. Recent literature suggests that “decoupling” the extraction logic into independent modules allows better fault tolerance. However, a major challenge remains in efficiently managing these decoupled modules without a centralized orchestration layer that can handle task scheduling and error recovery in real time.

Schema Heterogeneity and NoSQL Solutions

A widely cited problem in academic data engineering is “Schema Drift,” where different data sources provide information in wildly different formats. Traditional Relational Databases (RDBMS) require rigid table structures that fail when a new data field is introduced. Comparative studies have shown that NoSQL databases, specifically document stores such as MongoDB, offer the required horizontal scalability and schema flexibility to store diverse educational datasets without compromising data integrity [9].

Parallelism and Resource Orchestration

Although parallel processing is a well-established concept, its specific application in web ingestion is complex because of rate-limiting and IP-blocking by target servers. Current research emphasizes the need for an “Orchestrator” that can balance the load between multiple extraction tasks, such as college details, course fees, and exam updates, ensuring that high-speed ingestion does not compromise the stability of the host environment [10].

PROPOSED METHODOLOGY

The proposed framework was designed to overcome the limitations of sequential data extraction by implementing a decentralized hierarchical architecture. As shown in the system overview (Figure 1), the methodology is divided into three core functional layers:

1. *The master orchestration layer*: At apex of the hierarchy is the master controller, which serves as the central command center. Unlike traditional scripts that execute logic internally, this orchestrator manages the lifecycles of external processes. It handles task scheduling, determines execution priorities (e.g., time-sensitive entrance exam updates over static college profiles), and monitors system health. By tracking the CPU and RAM usage, the master prevents “memory leaks” by ensuring that the number of concurrent browser instances does not exceed the hardware’s threshold [11].
2. *Decoupled extraction modules (The Slave Layer)*: The core extraction logic is broken down into independent, domain-specific sub-modules. Each module is a standalone script optimized for a specific data entity, such as “Institute Information,” “Fee Structure,” or “Admission Eligibility.” Because these modules are decoupled, they can be updated or debugged individually without affecting the rest of the system design. This modularity is the cornerstone of the framework’s flexibility, allowing it to adapt to different website architectures.
3. *High-concurrency via headless automation*: To achieve industrial-grade speed, the framework utilizes Playwright for browser automation. The system initiates multiple “Headless” browser contexts in parallel. Unlike “Headed” browsers that render a user interface, headless instances are significantly faster and use fewer resources [12]. This allows the framework to interact with dynamic, JavaScript-heavy elements (such as pop-ups and lazy-loading tables) across dozens of different websites at the exact same time, effectively multiplying the data ingestion rate.

4. *Error handling and fault isolation*: A critical feature of this methodology is its “Isolated Failure” logic. In traditional sequential systems, a single 404 error or a changed HTML tag on one website would crash the entire pipeline. In our proposed framework, each extraction module runs in its own sandbox environment. If one module fails, the Master Orchestrator catches the exception, logs the error for later review, and allows all other modules to continue their work. This ensures that the master dataset is always partially updated, even if some are temporarily unavailable.
5. *Data normalization and NoSQL synchronization*: Before the data reaches the storage layer, they undergo a transformation process. This involved stripping HTML artifacts, normalizing currency formats, and rephrasing the gathered content to maintain a consistent tone. The processed data were then synchronized with MongoDB. We chose a NoSQL document store specifically to handle “Schema Drift”—a phenomenon where different websites provide different levels of details. MongoDB’s JSON-like structure allows the system to store varying data fields without requiring a rigid, predefined table structure, making it the ideal choice for heterogeneous educational data.

RESULT AND DISCUSSION

The Performance of the Proposed

The Hierarchical Orchestration Framework was evaluated based on execution speed, data integrity, and system stability. The following observations highlight the improvements over traditional sequential ingestion methods.

1. *Significant reduction in execution time*: The Primary metric for success was the “Time-to-Sync.” By implementing parallel execution through the Master–Orchestrator, the system achieved a 70% reduction in the time required to update the central database. While a sequential script would take several hours to crawl multiple educational portals, our concurrent framework completed the same task in a fraction of the time by utilizing the server’s full processing power.
2. *Enhanced fault tolerance*: During testing, several target websites experienced downtime or layout changes. In a standard monolithic setup, these issues would crash into the entire process. However, our results showed that the “Isolated Failure” logic successfully allowed the system to skip the failing module and continue extracting data from other sources. This led to a 95% “up-time” for the ingestion pipeline, ensuring that the database remained updated, even when individual sources were unstable.
3. *Scalability and resource management*: The system demonstrated linear scaling capability. As more “Slave” modules were added to handle new data categories (such as scholarship information or placement records), the Master Controller successfully managed the increased load without a significant spike in memory usage. This confirms that the hierarchical approach is suitable for large-scale industrial applications, where data sources are constantly growing.
4. *Data consistency and schema flexibility*: By utilizing MongoDB, we observed that “Schema Drift” (changes in the type of data provided by websites) no longer caused database errors. The system successfully captured and stored varying data fields, ranging from simple text to complex nested tables, without requiring manual intervention. This flexibility significantly reduces the “Technical Debt” usually associated with maintaining large-scale web scrapers.
5. *Discussion of practical impact*: The results suggest that for modern EdTech platforms, the bottleneck is not the speed of the Internet but the architecture of the scraper. Moving away from “one-by-one” processing to a “Parallel–Master” model allows organizations to provide real-time updates to students, which is critical during high-traffic periods like entrance exam seasons or admission deadlines.

CONCLUSION

The development and implementation of Hierarchical Orchestration Frameworks represent a significant departure from traditional high-latency data ingestion methods. By shifting the architecture from a monolithic, sequential model to a modular, “Master–Slave” configuration, we have demonstrated that industrial-scale web scraping can be both resilient and highly efficient. The results

clearly indicate that parallelism, when managed by a centralized orchestrator, not only drastically reduces the processing time but also provides a necessary layer of fault tolerance that is often missing in standard ETL pipelines.

A key takeaway from this study is the critical role of NoSQL integration in handling the inherent instability of web data. The use of MongoDB allowed the system to adapt to “Schema Drift” effortlessly, ensuring that the diversity of educational information—from course fees to exam schedules—remained intact without the need for constant manual reconfiguration. This flexibility, combined with the high-speed execution provided by Playwright, offers a scalable blueprint for any data-driven platform operating in a complex, rapidly changing digital environment.

Looking forward, there is significant potential to further enhance this framework by incorporating machine learning algorithms to predict website structure changes before they cause module failure. Additionally, the integration of automated proxy rotation would further improve the system’s anonymity and prevent rate-limiting during peak ingestion periods. Ultimately, this project proves that a well-orchestrated, decentralized approach is essential for maintaining the data integrity and real-time responsiveness required by modern educational technology platforms.

FUTURE SCOPE

Although the current framework successfully addresses the primary challenges of speed and modularity, several avenues for further enhancement could redefine the efficiency of automated data ingestion.

Integration of Self-Healing AI

One of the most promising future developments is the implementation of a “self-healing” layer. By utilizing Large Language Models (LLMs) or Computer Vision, the Master Orchestrator can be trained to recognize when a target website’s layout has changed. Instead of simply logging an error, the system can autonomously re-map the new HTML tags in real time, allowing for uninterrupted data extraction without human intervention.

Advance Proxy Rotation and Anti-Bot Measures

As web security becomes more sophisticated, future iterations of this framework will focus on incorporating dynamic proxy rotation and “human mimicry” algorithms. By rotating IP addresses and varying interaction patterns (such as mouse movements and scrolling speeds), the system can avoid rate limiting and “IP-blocking” by target servers, ensuring long-term access to highly protected data sources.

Real-Time Sentiment and Trend Analysis

Currently, the system focuses on structured data, such as fees and dates. Future expansion could involve a natural language processing (NLP) module that scrapes student reviews and social media mentions. This would allow the platform to provide not just factual data, but also “sentiment scores” for colleges and exams, offering a more holistic view to the end-user.

Cloud Native Serverless Deployment

To achieve “infinite” scalability, the next phase of development involves migrating the “Slave” modules to a serverless environment (such as AWS Lambda or Google Cloud Functions). This would allow the system to spin up hundreds of extractors simultaneously on demand and shut them down immediately after the task is complete, drastically reducing server costs and energy consumption.

REFERENCES

1. Rucco C, Longo A, Saad M. Enhancing data ingestion efficiency in cloud-based systems: a design pattern approach. *Data Sci Eng.* 2026 Jun;11(2):287–302.
2. Talakola S. Automated end to end testing with Playwright for React applications. *Int J Emerg Res Eng Technol.* 2024 Mar 30;5(1):38–47.

3. MongoDB Documentation Team. What is MongoDB? [Online]. MongoDB; 2026. Available from: <https://www.mongodb.com/docs/manual/>
4. de Lima Melo FF, Ferreira F. Performance analysis of NoSQL databases with large volumes of open educational data. *Int J Comput Appl*. 2024;174(29):9–17.
5. Ma J. A high performance computing web search engine based on big data and parallel distributed models. *Informatica*. 2024;48(20):27–38.
6. Bezawada N, Rajagopal AR, Swaminathan A, Smrithy GS, Elakkiya R. Optimizing data extraction techniques in the automation industry. In: *International Conference on Hybrid Intelligent Systems*; 2023 Dec 11. p. 89–98. Cham: Springer Nature Switzerland.
7. Eyzenakh DS, Rameykov AS, Nikiforov IV. High performance distributed web-scraper. *Tr Inst Sist Program RAN*. 2021;33(3):87–100.
8. Chaganti S. Automated schema enforcement and stateless deduplication in distributed data pipelines: a contract-driven approach. In: *SoutheastCon 2026*; 2026 Feb 20. p. 1–5.
9. Richardson L, Ruby S. *RESTful web services*. Sebastopol (CA): O'Reilly Media; 2008 Dec 17.
10. Chodorow K. *MongoDB: the definitive guide*. Sebastopol (CA): O'Reilly Media; 2013.
11. Lewis J, Fowler M. *Microservices: a definition of this new architectural term* [Online]. MartinFowler.com. 2014 Mar 25. Available from: <https://martinfowler.com/articles/microservices.html>
12. Roth J, Duan Y, Mahner FP, Kaniuth P, Wallis TS, Hebart MN. Ten principles for reliable, efficient, and adaptable coding in psychology and cognitive neuroscience. *Commun Psychol*. 2025 Apr 15;3(1):62.