Distributed Video Transcoding Engine

January 16, 2026

1 Overview

The goal of this project is to optimize the time-intensive process of video transcoding by utilizing parallel processing. Standard transcoding is typically a single-threaded, CPU-bound task. This system bypasses that limitation by segmenting a video into smaller parts and distributing the workload across multiple isolated Docker environments.

2 System Architecture

The system follows a **Master-Worker (Orchestrator-Consumer)** pattern, managed through containerization to ensure scalability and environment consistency.

- **Orchestrator (Master):** A Python-based controller that uses FFmpeg to split the source video into 5-second segments and pushes tasks to a Redis queue.
- **Message Broker (Redis):** Acts as a non-blocking task queue, ensuring multiple workers pull tasks asynchronously without duplication.
- **Workers (Docker Replicas):** Multiple isolated containers running Python. Each worker pulls a task, transcodes the segment to 720p, and saves it to the output directory.
- **Final Merger:** A post-processing step that concatenates the transcoded segments back into a single video file.

3 Technology Stack

Component	Technology	Role
Logic	Python 3	Orchestration and worker logic
Media Engine	FFmpeg	Splitting, transcoding, and merging
Infrastructure	Docker & Compose	Containerization and scaling
Message Broker	Redis	Distributed task management

Table 1: System Technology Stack

4 Implementation Details

4.1 Parallelism and Scaling

By using Docker Compose `replicas`, the system can spin up any number of workers. On the testing hardware (i5 8th Gen), 3 workers were utilized to effectively saturate the multi-core architecture.

4.2 Task Handling

The use of Redis `BLPOP` ensures atomic task distribution. Workers remain idle until a new task appears in the queue, making the system highly resource-efficient.

5 Performance Analysis

During testing, the distributed mechanism showed a significant reduction in encoding time compared to linear processing. By saturating 3 workers simultaneously, total encoding time was reduced by approximately 60-70% relative to a single-threaded process.

6 Conclusion

This project demonstrates how DevOps tools like Docker and Redis can solve computational bottlenecks. The system provides a scalable foundation that can be easily expanded to a cloud environment by increasing the worker replica count.