⌚Overwatch

Executive Summary

To address user concerns regarding performance and scalability challenges in Fossology's scheduler (Daemon 👷), the adoption of an **off-the-shelf (OTS)** scheduler solution is proposed. This solution aims to enhance maintainability, boost efficiency, and improve overall throughput, ensuring the platform can meet growing demands effectively.

Problem Statement

Daemon 👷faces significant challenges that impact its performance, maintainability, and scalability which includes:

1. **Limitations Of Language:** The scheduler, written in C, lacks built-in exception handling, making debugging and maintenance complex and error-prone.
2. **Flaws in Queue Design:** The linear queue design does not allow fine-grained control over job execution. Mutually exclusive jobs block other unrelated tasks, leading to inefficiencies.
3. **Scalability Bottlenecks:** The blocking nature of I/O operations limits resource utilization, particularly during high workloads, resulting in task delays and reduced throughput.

How We Identified the Problem:

- **Long-standing Design Issues:** The scheduler has not been updated for years, leading to technical debt and growing inefficiencies.
- **User Feedback and Support Requests:** Users and contributors reported issues with task execution delays and mismanagement of agent exclusivity, highlighting inefficiencies in the scheduler design.

Proposed Solution:

The solution involves a comprehensive scheduler overhaul with the following key objectives:

- **Change in Programming Language:** Transitioning to modern, lightweight frameworks that support exception handling, concurrency, and parallelism to improve maintainability and robustness.
- **Redesign of Queue:** Adopting a bucketed queue structure, where each upload maintains its own priority queue, and a global queue handles overarching tasks such as maintenance and delegation. A round-robin algorithm will traverse buckets, prioritize jobs, and ensure compliance with host limits. Exclusive agents will be routed to the global queue, preventing bottlenecks.
- **Enhancements in Scalability:** Optimizing I/O operations to maximize resource utilization and throughput, ensuring efficient performance during high demand.

User Stories:

Fossology's stakeholders require a scheduler that is **efficient, maintainable, and scalable to meet evolving needs.** The scheduler must prevent tasks for individual uploads from blocking unrelated jobs, ensuring smooth and uninterrupted workflows. It should also **effectively manage exclusive agents and prioritize tasks to maximize resource utilization**. To enhance maintainability, implementing the scheduler in a modern programming language is essential, **simplifying debugging and reducing long-term technical debt.** Additionally, a redesigned queue structure with upload-specific buckets and a global queue is needed to **support localized and overarching operations**. Leveraging contemporary scheduling libraries will further align the system with current technologies and ensure compatibility with future demands, enabling Fossology to scale effectively while addressing user and operational requirements.

Goals

- Ensure tasks for individual uploads do not block unrelated jobs, reducing workflow delays.
- Manage exclusive agents and prioritizing tasks.
- Simplify debugging and increase maintainability by implementing programming language.

- Redesign the queue structure to include upload-specific buckets and a global queue for better flexibility and scalability.

Non-Goals

- Rewritten the entire Fossology platform or unrelated components.
- Adding new features unrelated to scheduling, such as UI enhancements or additional agent functionalities.
- Support legacy systems or configurations that are incompatible with modern scheduling frameworks.
- Addressing peripheral performance issues unrelated to the scheduler, such as database or network optimizations.

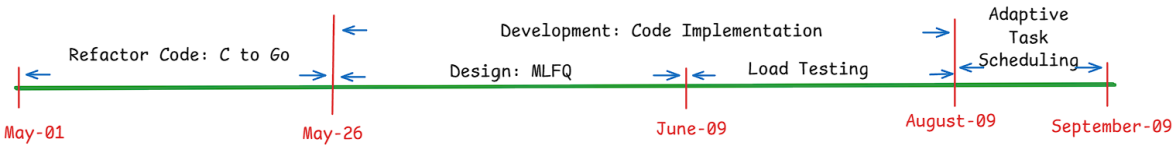
Key Metrics

1. **Task Throughput**: Increase the number of tasks processed per item
2. **Queue Efficiency:** Improve average task wait time by 50% by implementing upload-specific queues
3. **Resource Utilisation:** Achieve 90% resource utilization during peak loads without task-blocking.
4. **CSAT:** Increase user satisfaction scores related to scheduler performance by 20% within six months of deployment.

Rollback Criteria

If the new scheduler does not improve task throughput per item, **reduce task wait time by 50%**, **achieve 90% resource utilization** during peak loads, or **increase CSAT by 20%** within six months, we will initiate a rollback. Formal evaluations at three and six months will guide the decision, with a sprint allocated to disable the new scheduler if necessary.

Timelines



Design & Mocks

- **Queue Architecture Diagram**
- **Scheduler Workflow Chart**
- **Agent Interaction Mock**

Technical Concerns & Outstanding Questions

Concurrency Challenges: How to minimize race conditions when scaling the system for larger datasets.

Integration of Machine Learning Models:

- Feasibility and complexity of implementing ML for real-time resource allocation predictions.

Offline Capabilities:

- Maintaining task execution consistency without live API data in unstable network environments.

Cache Consistency:

- Ensuring cache invalidation strategies are robust to avoid stale data affecting scheduling decisions.

Appendix

Future Enhancements:

- Explore asynchronous messaging queues for reliable alert systems.
- Develop offline-capable modules to improve resilience in disconnected environments.