MRVA component interconnections

Michael Hohn

Technical Report 20250524

Contents

1 Overview

The MRVA system is organized as a collection of services. On the server side, the system is containerized using Docker and comprises several key components:

- **Server**: Acts as the central coordinator.
- **Agents**: One or more agents that execute tasks.
- **RabbitMQ**: Handles messaging between components.
- **MinIO**: Provides storage for both queries and results.
- **HEPC**: An HTTP endpoint that hosts and serves CodeQL databases.

The execution process follows a structured workflow:

1. A client submits a set of queries $\mathscr{Q}$ targeting a repository set $\mathscr{R}$.
2. The server enqueues jobs and distributes them to available agents.
3. Each agent retrieves a job, executes queries against its assigned repository, and accumulates results.
4. The agent sends results back to the server, which then forwards them to the client.

This full round-trip can be expressed as:

$$\text{Client} \xrightarrow{\mathscr{Q}} \text{Server} \xrightarrow{\text{enqueue}} \text{Queue} \xrightarrow{\text{dispatch}} \text{Agent} \xrightarrow{\mathscr{Q}(\mathscr{R}_i)} \text{Server} \xrightarrow{\mathscr{Q}(\mathscr{R}_i} \text{Client} \tag{1}$$

2 Symbols and Notation

We define the following symbols for entities in the system:

Concept	Symbol	Description
Client	C	The source of the query submission
Server	S	Manages job queue and communicates results back to the client
Job Queue	Q	Queue for managing submitted jobs
Agent	α	Independently polls, executes jobs, and accumulates results
Agent Set	A	The set of all available agents
Query Suite	$\mathscr{Q}$	Collection of queries submitted by the client
Repository List	$\mathscr{R}$	Collection of repositories
i-th Repository	$\mathscr{R}_i$	Specific repository indexed by i
j-th Query	$\mathscr{Q}_j$	Specific query from the suite indexed by j
Query Result	$r_{i,j,k_{i,j}}$	$k_{i,j}$-th result from query j executed on repository i
Query Result Set	$\mathscr{R}_i^{\mathscr{Q}_j}$	Set of all results for query j on repository i
Accumulated Results	$\mathscr{R}_i^{\mathscr{Q}}$	All results from executing all queries on $\mathscr{R}_i$

3 Full Round-Trip Representation

The full round-trip execution, from query submission to result delivery, can be summarized as:

$$C \xrightarrow{\mathscr{Q}} S \xrightarrow{\text{enqueue}} Q \xrightarrow{\text{poll}} \alpha \xrightarrow{\mathscr{Q}(\mathscr{R}_i)} S \xrightarrow{\mathscr{R}_i^{\mathscr{Q}}} C$$

- $C \to S$: Client submits a query suite $\mathscr{Q}$ to the server.
- $S \to Q$: Server enqueues the query suite $(\mathscr{Q}, \mathscr{R}_i)$ for each repository.
- $Q \to \alpha$: Agent α polls the queue and retrieves a job.
- $\alpha \to S$: Agent executes the queries and returns the accumulated results $\mathscr{R}_i^{\mathscr{Q}}$ to the server.
- $S \to C$: Server sends the complete result set $\mathscr{R}_i^{\mathscr{Q}}$ for each repository back to the client.

4 Result Representation

For the complete collection of results across all repositories and queries:

$$\mathscr{R}^{\mathscr{Q}} = \bigcup_{i=1}^{N} \bigcup_{j=1}^{M} \left\{ r_{i,j,1}, r_{i,j,2}, \ldots, r_{i,j,k_{i,j}} \right\}$$

where:

- N is the total number of repositories.
- M is the total number of queries in $\mathscr{Q}$.
- $k_{i,j}$ is the number of results from executing query $\mathscr{Q}_j$ on repository $\mathscr{R}_i$.

An individual result from the i-th repository, j-th query, and k-th result is:

$$r_{i,j,k}$$

$$C \xrightarrow{\mathscr{Q}} S \xrightarrow{\text{enqueue}} Q \xrightarrow{\text{dispatch}} \alpha \xrightarrow{\mathscr{Q}(\mathscr{R}_i)} S \xrightarrow{r_{i,j}} C$$

Each result can be further indexed to track multiple repositories and result sets.

5 Graph Extraction from Log Table

Assume we have a structured event log represented as a set of tuples.

Event Log Structure

Let

$$\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$$

be the set of all events, where each event

$$t_i = (id_i, \tau_i, a_i, e_i, q_i, r_i, c_i)$$

consists of:

- id_i: unique event ID
- τ_i: timestamp
- a_i: actor (e.g., "agent_alpha1")
- e_i: event type (e.g., "enqueue", "execute")
- q_i: query ID
- r_i: repository ID
- c_i: result count (may be $\perp$ if not applicable)

Let

$$\mathcal{G} = (V, E)$$

be a directed graph constructed from $\mathcal{T}$, with vertices V and edges E.

Graph Definition

$$V = \{id_i \mid t_i \in \mathcal{T}\}$$
$$E \subseteq V \times V$$

Edges capture temporal or semantic relationships between events.

Construction Steps

1. Partition by Job Identity Define the set of job identifiers:

$$J = \{(q, r) \mid \exists i : q_i = q \wedge r_i = r\}$$

Then for each $(q, r) \in J$, define:

$$\mathcal{T}_{q,r} = \{t_i \in \mathcal{T} \mid q_i = q \wedge r_i = r\}$$

2. Sort by Time Order each $\mathcal{T}_{q,r}$ as a list:

$$\mathcal{T}_{q,r} = [t_{i_1}, t_{i_2}, \ldots, t_{i_k}] \quad \text{such that } \tau_{i_j} < \tau_{i_{j+1}}$$

3. Causal Edges Define within-job edges:

$$E_{q,r} = \{(id_{i_j}, id_{i_{j+1}}) \mid 1 \leq j < k\}$$

4. Global Causal Graph Take the union:

$$E_{\text{causal}} = \bigcup_{(q,r)\in J} E_{q,r}$$

5. Semantic Edges (Optional) Define semantic predicates such as:

$$\text{pulls}(i,j) \iff e_i = \text{enqueue} \wedge e_j = \text{pull} \wedge q_i = q_j \wedge r_i = r_j \wedge \tau_i < \tau_j \wedge a_i = \text{server} \wedge a_j = \text{agent}$$

Then:

$$E_{\text{semantic}} = \{(id_i, id_j) \mid \text{pulls}(i,j)\}$$

Final Graph

$$V = \{id_i \mid t_i \in \mathcal{T}\}$$
$$E = E_{\text{causal}} \cup E_{\text{semantic}}$$

Notes

- This construction is generic: the log store $\mathcal{T}$ may come from a database, file, or tuple-indexed dictionary.
- Each semantic edge rule corresponds to a logical filter/join over $\mathcal{T}$.
- The construction is schema-free on the graph side and can be recomputed on demand with different edge logic.