Operations and Management Area Working Group C. Zhou, Ed.
Internet-Draft Y. Mo
Intended status: Informational H. Liu
Expires: 16 November 2025 Y. Pan
Huazhong University of Science and Technology
15 May 2025

Cross-Domain Cloud-Native Resource Orchestration Framework with Dynamic Weight-Based Scheduling
draft-zhou-crosscloud-orchestration-02

Abstract

The Distributed Resource Orchestration and Dynamic Scheduling (DRO-DS) standard in cross-domain cloud-native environments aims to address the challenges of resource management and scheduling in multi-cloud architectures, providing a unified framework for efficient, flexible, and reliable resource allocation. As enterprise applications scale, the limitations of single Kubernetes clusters become increasingly apparent, particularly in terms of high availability, disaster recovery, and resource optimization. To address these challenges, DRO-DS introduces several innovative technologies, including dynamic weight-based scheduling, storage-transmission-compute integration mechanisms, follow-up scheduling, real-time monitoring and automated operations, as well as global views and predictive algorithms.

Table of Contents

1. Introduction

The evolution of cloud-native computing has precipitated the emergence of Kubernetes as a predominant orchestration framework for containerized workloads, a paradigm substantiated by its widespread adoption in enterprise environments. This technological progression, however, reveals inherent architectural constraints when applied to singular administrative domains, manifesting principally in suboptimal fault tolerance mechanisms, inadequate disaster recovery protocols, and inefficient resource allocation strategies under scaled operational demands.

Contemporary infrastructure paradigms demonstrate increasing adoption of multi-cloud deployment models, a trend driven by their inherent advantages in operational expenditure optimization, geospatial fault domain distribution, and compliance-driven environmental segregation. These hybrid architectures nevertheless introduce novel systemic complexities. Current implementations typically exhibit network segmentation patterns where Kubernetes clusters operate within discrete subnet boundaries, engendering resource fragmentation that fundamentally constrains dynamic workload redistribution while inducing cross-domain load asymmetry.

The Kubernetes ecosystem's response to these challenges materialized through KubeFed V2, a federated scheduling mechanism designed for multi-cluster coordination. Academic evaluations and industry implementation reports, however, identify persistent limitations in its architectural implementation. Notable deficiencies include the reliance on predetermined weighting coefficients for resource distribution, incomplete integration patterns for stateful workload management, and constrained telemetry capabilities for real-time cluster state observation.

To address these shortcomings, there is an urgent need for a new cross-domain scheduling engine capable of dynamically managing and optimizing resources across multiple domains, providing robust support for stateful services, and offering comprehensive real-time monitoring and automation capabilities. This document proposes the Distributed Resource Orchestration and Dynamic Scheduling (DRO-DS) standard for cross-domain cloud-native environments, designed to meet these requirements. This document normatively references [RFC5234] and provides additional information in [KubernetesDocs] and [KarmadaDocs].

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Scope

This standard provides a unified framework for distributed resource management across cloud service providers, regions, and heterogeneous computing environments. It is primarily applicable to scenarios such as cross-domain resource orchestration in multi-cloud architectures (e.g., hybrid deployments on AWS/Azure/GCP), distributed resource scheduling (e.g., resource coordination between eastern and western data centers), unified management of heterogeneous computing environments (e.g., CPU/GPU/FPGA hybrid architectures), and cross-domain high-availability deployment of stateful services (e.g., databases, message queues). The technical implementation boundary is strictly limited to resource abstraction layer scheduling strategies and does not involve specific details of underlying network infrastructure. It adheres to the principle of cloud service provider neutrality and does not mandate specific vendor implementation solutions.

The intended audience for this standard includes multi-cloud architecture designers, cloud-native platform developers, and distributed system operations engineers. As an enhanced extension component of existing single-cluster schedulers (e.g., the Kubernetes default scheduler), this standard does not replace basic scheduling functionalities. Instead, it introduces innovative mechanisms such as dynamic weight-based scheduling and storage-transmission-compute integration to achieve collaborative optimization of cross-domain resources. The technical specifications focus on control plane architecture design, scheduling process standardization, and API interface definitions, while maintaining openness to specific implementation technologies on the data plane.

3. Terminology

3.1. Definitions

* *Cross-Domain*: Refers to multiple independent Kubernetes clusters or cloud environments.

* *Distributed Resource Orchestration (DRO)*: The process of managing and coordinating resources across multiple domains.

* *Storage-Transmission-Compute Integration (STCI)*: A method for unified management of storage, data transmission, and computing resources.

* *Resource Water Level (RWL)*: A metric representing the proportion of current resource usage relative to total available resources.

* *Global View (GV)*: A comprehensive overview of all domain resources and their states.

* *Follow-Up Scheduling (FS)*: A scheduling mechanism that ensures consistency and efficiency of stateful services across domains.

3.2. Abbreviation

* *Kubernetes (K8s)*: An open-source container orchestration platform.

* *Cloud-Native*: Applications and services specifically designed to leverage cloud computing platforms.

* *Dynamic Scheduling (DS)*: A scheduling mechanism that adapts to real-time resource availability and demand.

* *Application Programming Interface (API)*: A set of rules and protocols for building and interacting with software applications.

* *Key-Value Store (KV Store)*: A type of database that stores, retrieves, and manages data using a simple key-value method.

* *Predictive Algorithm (PA)*: An algorithm that forecasts future resource demands based on historical data and current trends.

* *Stateful Service (SS)*: Services such as databases and caches that maintain state between client interactions.

* *Real-Time Monitoring (RTM)*: Continuous real-time monitoring of system health and resource usage.

4. Overview

In this section, we introduce the challenges in this field and the functional requirements of DRO-DS.

4.1. BACKGROUND AND CHALLENGES

The cross-domain deployment of cloud-native technologies is undergoing a paradigm shift from single-domain to multi-cloud hybrid architectures. Industry practices indicate that enterprises commonly face systemic performance degradation caused by cross-cloud resource scheduling, exposing deep-seated design limitations of traditional scheduling mechanisms in dynamic heterogeneous environments. The fundamental contradictions are reflected in three aspects:

4.1.1. RESOURCE FRAGMENTATION CHALLENGES

* Differences in resource abstraction models across heterogeneous cloud platforms create scheduling barriers. Fragmentation in virtual machine specifications, storage interfaces, and network QoS policies makes it difficult to construct a global resource view.

* Storage locality constraints and spatiotemporal mismatches in computing resource distribution lead to simultaneous resource idling and contention.

* Bandwidth fluctuations and cost sensitivity in cross-domain network transmission significantly increase the complexity of scheduling strategies.

4.1.2. SCHEDULING LATENCY BOTTLENECKS

* Periodic polling-based state awareness mechanisms struggle to capture instantaneous load changes, resulting in decision biases during traffic burst scenarios.

* The cumulative delay effect of cross-domain communication in the control plane causes policy synchronization lags, which worsen nonlinearly in large-scale network domain scenarios.

* Static resource allocation strategies cannot adapt to the diurnal fluctuations of workloads, leading to resource mismatches.

4.1.3. OPERATIONAL COMPLEXITY DILEMMAS

* Semantic differences in heterogeneous monitoring systems reduce the efficiency of root cause analysis.

* The cross-domain extension of service dependency chains results in fault propagation paths with mesh topology characteristics.

* Implicit errors caused by environmental configuration drift are exponentially amplified in cross-domain scheduling scenarios.

4.2. Function Requirements of DRO-DS

DRO-DS should support the following functionalities:

1. *Unified Abstraction and Modeling of Cross-Domain Resources*: The system must establish a standardized resource description framework, supporting unified semantic mapping of resources such as virtual machines, storage, and networks in multi-cloud heterogeneous environments. This eliminates differences in resource specifications and interface policies across cloud platforms, enabling discoverability, measurability, and collaborative management of global resources, thereby addressing scheduling barriers caused by resource fragmentation.

2. *Dynamic Weight Elastic Scheduling Mechanism*: Based on real-time domain resource water levels, load states, and network topology data, dynamically calculate scheduling weight coefficients for each domain. Automatically adjust resource allocation directions based on task priorities and business policies to achieve balanced distribution in high-load scenarios and resource aggregation in low-load scenarios, overcoming the latency bottlenecks of static scheduling strategies.

3. *Topology-Aware Scheduling for Stateful Services*: For stateful services such as databases and message queues, integrate network latency detection, storage locality constraint analysis, and service dependency relationship modeling to intelligently select the optimal deployment domain and provide smooth migration strategies, ensuring service continuity, data consistency, and fault recovery capabilities in cross-domain scenarios.

4. *Integrated Storage-Transmission-Compute Collaborative Scheduling*: Through data hotspot identification, hierarchical caching strategies, and incremental synchronization optimization, achieve tightly coupled decision-making for storage resource distribution, data transmission paths, and computing task scheduling, minimizing cross-domain data transfer overhead and improving resource utilization efficiency.

5. *Cross-Domain Intelligent Monitoring and Automated Operations*: Build a unified collection and standardized transformation layer for heterogeneous monitoring data, support topology modeling and root cause analysis of service dependency chains, and automatically trigger operations such as scaling and failover based on predefined policies, reducing the complexity of manual intervention.

6. *Elastic Scaling and Cost-Aware Optimization*: Integrate historical load pattern analysis and multi-cloud cost models to achieve predictive resource pre-allocation and cost-performance-optimized scheduling strategies, support elastic resource provisioning for burst traffic, and optimize cross-cloud costs, avoiding resource mismatches and waste.

7. *Cloud-Native Ecosystem Compatibility*: Maintain API compatibility with mainstream container orchestration systems such as Kubernetes, support Custom Resource Definition (CRD) extensions and cross-domain federation management standards, avoid vendor lock-in, and reduce adaptation costs for existing architectures.

4.3. System Interaction Model

4.3.1. Control Plane Model

In multi-domain environments, to effectively manage multiple domains, a dedicated control plane is necessary to handle domain joining/eviction, state management, and application scheduling. The control plane is logically separated from the domains (data plane) where applications actually run. Depending on enterprise needs and scale, the control plane in a DRO-DS-based management/control architecture

can adopt two deployment modes: control plane with dedicated domain resources and control plane sharing domain resources with the data plane.

4.3.1.1. Control Plane with Dedicated Domain Resources

In this mode, control plane components exclusively occupy one or more dedicated domains as "control domains" for executing all multi-domain management decisions. These control domains do not run actual business applications but focus on managing the state and scheduling of other worker domains. Control domains can ensure high availability through election mechanisms, avoiding single points of failure. Worker domains contain multiple "worker nodes," each running actual business applications. As shown in Figure 1, the control plane and data plane are physically isolated, ensuring system stability and security. Deploying multiple control domains achieves high availability of the control plane, preventing system-wide failures due to a single control domain failure. This deployment mode is suitable for complex, large-scale multi-domain environments with high isolation and stability requirements.

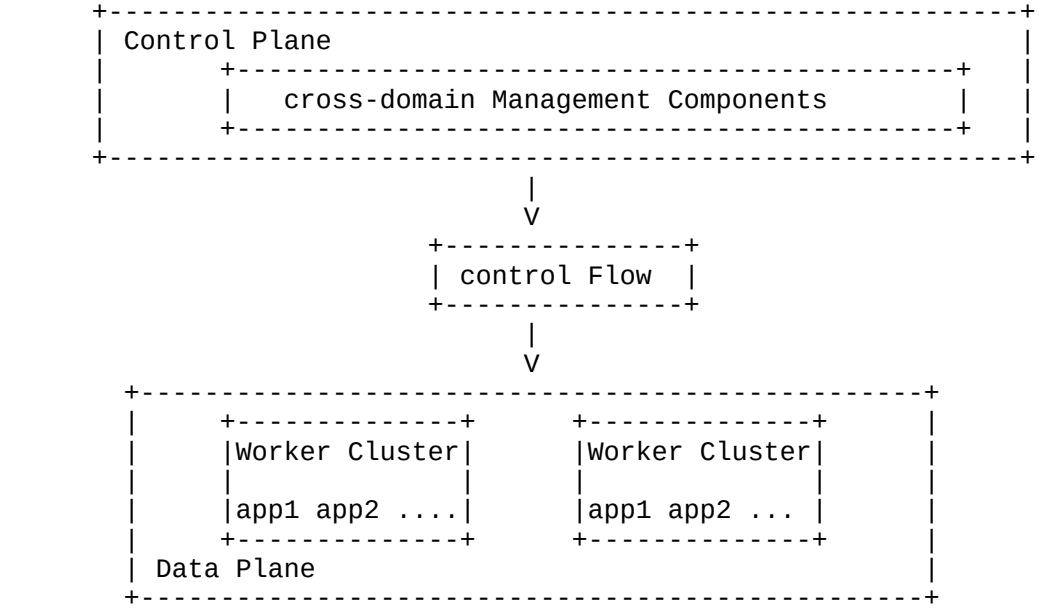

Figure 1: Control Plane with Dedicated Domain Resources

4.3.1.2. Control Plane Sharing Domain Resources with Data Plane

In this mode, control plane components share the same general domain with business applications. Control plane components determine master-slave relationships through election mechanisms, with the master control plane responsible for management decisions across all general domains. This deployment approach simplifies infrastructure and reduces resource costs but may lead to mutual interference between the control plane and data plane. As shown in Figure 2, this mode does not require additional dedicated control domains, resulting in lower overall resource usage. It is suitable for small-scale multi-domain environments with relatively simple deployment and maintenance.

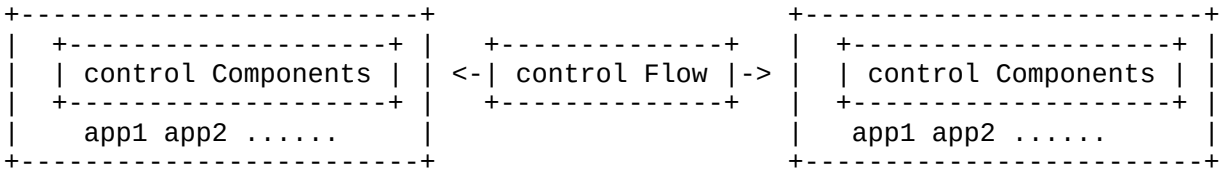

Figure 2: Control Plane Sharing Domain Resources with Data Plane

4.3.2. Network Model

In the network model, multiple domains (whether based on the same or different cloud providers) typically operate in network isolation, each within its own subnet. Therefore, ensuring network connectivity becomes a critical issue in multi-domain collaboration. The network connectivity between domains depends on specific usage scenarios. For example, in tenant isolation scenarios, strict network isolation is required between domains of different tenants to ensure security; in high availability or elastic scaling scenarios, business applications may require east-west network access, ensuring that applications can communicate with each other regardless of the domain they run in. Additionally, the multi-domain control plane must be able to connect to all domains to implement management decisions. To achieve inter-domain connectivity, the following two approaches can be used.

4.3.2.1. Gateway Routing

Gateway routing is a method of achieving cross-domain communication by deploying gateways in each domain. Specifically, gateways are responsible for forwarding packets from one domain to the target address in another domain. This approach is similar to the multi-network model in service meshes (e.g., Istio), where cross-domain business application network activities are forwarded by gateways,

ensuring service-to-service communication.

4.3.2.2. Overlay Network Overlay

An overlay network is a method of constructing a virtual network using tunneling technology, enabling multiple domains to be logically in the same virtual subnet despite being physically located in different clouds or VPCs, thereby achieving direct network connectivity. The core idea of an overlay network is to build tunnels over the Layer 3 network (IP network) to transmit Layer 2 packets (e.g., Ethernet frames), forming a flat virtual network.

4.3.3. Service Discovery and Governance

In this standard, a cross-domain service (Multi-Cluster Service) refers to a collection of service instances spanning multiple domains, and its management and discovery mechanisms follow the definitions in [KEP-1645]. Cross-domain services allow applications to perform consistent service registration, discovery, and access across multiple domains, ensuring seamless operation of business applications regardless of their domain location. Service registration information is written to the KV Store via the API Server, and the scheduler makes cross-domain service routing decisions based on the global service directory, with the monitoring system synchronizing the health status of services in each domain in real time.

5. Architecture

5.1. Logic Architecture

The architecture design references [KarmadaDocs]. This document provides a reference functional architecture for DRO-DS, as shown in Figure 1.

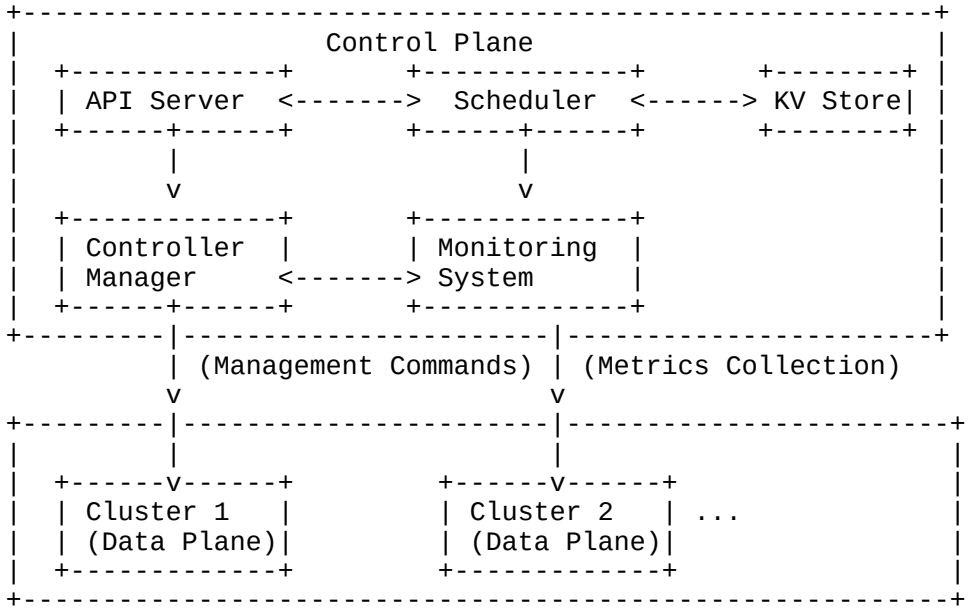

Figure 3: Logic Architecture Of DRO-DS

The DRO-DS architecture aims to provide a unified framework for resource management and scheduling across multiple domains in cloud-native environments. The key components of the architecture include:

* *DRO-DS API Server*: The DRO-DS API Server serves as the central hub for receiving and distributing scheduling requests. It communicates with other system components, coordinates commands, and ensures that tasks are properly scheduled and executed. The API Server supports both synchronous and asynchronous communication, allowing flexible integration with various tools and systems.

* *DRO-DS Scheduler*: The DRO-DS Scheduler is responsible for the core resource scheduling tasks. It integrates the characteristics of resources across three dimensions - network, storage, and computing power - to construct a multi-modal network resource scheduling framework. It intelligently allocates these resources to ensure optimal performance and resource utilization. The scheduler employs the Storage-Transmission-Compute Integration (STCI) method, tightly coupling storage, data transmission, and computing to minimize unnecessary data movement and improve data processing efficiency.

The scheduler uses a dynamic weight-based scheduling mechanism, where the weight assigned to each domain is determined by its Resource Water Level (RWL) - a metric that reflects the proportion of current resource usage relative to total available resources. This dynamic approach enables the scheduler to adapt to changes in resource availability and demand, ensuring balanced resource allocation across domains.

* *DRO-DS Controller Manager*: The DRO-DS Controller Manager consists of a set of controllers for managing both native and custom resources. It communicates with the API Servers of individual domains to create and manage Kubernetes resources. The Controller Manager abstracts all domains into a unified Cluster-type resource, enabling seamless cross-domain resource and service discovery. This component also ensures consistent management of stateful services such as databases and caches across domains.

* *DRO-DS Monitoring System*: The DRO-DS Monitoring System provides real-time visibility into the health status and resource usage of all domains. It collects and analyzes metrics from each domain, offering comprehensive monitoring data to help operators quickly identify and resolve issues. The Monitoring System supports automated operations, automatically adjusting scheduling strategies based on real-time resource usage and system health. This ensures high availability and efficiency of the system even under changing conditions.

* *Key-Value Store (KV Store)*: The KV Store is a distributed key-value database that stores metadata about the system, ensuring consistency and reliability across all domains. It supports the Global View (GV) of the system, enabling the scheduler to make informed decisions based on a comprehensive understanding of resource availability and usage. The KV Store also facilitates the implementation of Predictive Algorithms (PA), which forecast future resource demands based on historical data and current trends.

6. API Specification

6.1. General Conventions

* *Authentication Method*: The Bearer Token authentication mechanism is used, with the request header including "Authorization: Bearer <jwt_token>".

* *Version Control Policy*: All interface paths start with "/api/v1/", and in the event of a subsequent version upgrade, the path changes to "/api/v2/".

* *Error Code Specification*: The error response body includes the following fields:

  - code: The error code (e.g., CLUSTER_NOT_FOUND).

  - message: A human-readable error description.

  - details: An error detail object.

Example error response:

```
{
  "code": "POLICY_CONFLICT",
  "message": "Policy name already exists",
  "details": {"policyName": "default-policy"}
}
```

6.2. Control Plane API

6.2.1. Cluster Management Interface

* *Interface Path*: /api/v1/clusters

* *Methods and Functions*:

  - GET: Retrieve a list of all registered domains.

  - POST: Register a new domain.

```
{
  #Request example
  "name": "gcp-production",
  "endpoint": "https://k8s.gcp.example",
  "capabilities": ["gpu", "tls-encryption"]
}
```

  - DELETE /{clusterName}：解除域注册。

6.2.2. Policy Management Interface

* *Interface Path*: /api/v1/propagationpolicies

* *Methods and Functions*:

  - POST: Create a resource propagation policy.

```
{
  #Request example
  "name": "cross-region-policy",
  "resourceType": "Deployment",
  "targetClusters": ["aws-us-east", "azure-europe"]
}
```

- PATCH /{policyName}: Update a policy using theapplication/ merge-patch+json format.

6.2.3. ResourcePlacementPolicy *Interface*

* *Interface Path*: /apis/multicluster.io/v1alpha1/ resourceplacementpolicies

* *Methods and Functions*:

  - GET: Retrieve a list of all resource placement policies or query a specific policy.

  - POST: Create a resource placement policy, defining the target domains for resource distribution.

  - DELETE /{policyName}: Delete a specified placement policy.

```
#POST creation example
apiVersion: multicluster.io/v1alpha1
kind: ResourcePlacementPolicy
metadata:
  name: webapp-placement
spec:
  resourceSelector:
    apiVersion: apps/v1
    kind: Deployment
    name: webapp
    namespace: production
  targetClusters:
    - selectorType: name  # Supports "name" or "label"
      values: ["cluster-east", "cluster-west"]
```

6.2.4. ClusterOverridePolicy Interface

* *Interface Path*: /apis/multicluster.io/v1alpha1/ clusteroverridepolicies

* *Methods and Functions*:

- GET: Retrieve a list of all override policies or query a specific policy.

- POST: Create an override policy for a specific domain, overriding the configuration of resources in that domain.

- DELETE /{policyName}: Delete a specified override policy.

```
  # POST creation example
    apiVersion: multicluster.io/v1alpha1
    kind: ClusterOverridePolicy
    metadata:
      name: gpu-override
    spec:
      resourceSelector:
        apiVersion: apps/v1
        kind: Deployment
        name: ai-model
        namespace: default
      overrides:
        - clusterSelector:
            names: ["gpu-cluster"]
          fieldPatches:
            - path: spec.template.spec.containers[0].resources.limits
              value: {"nvidia.com/gpu": 2}
```

6.2.5. Task Scheduling Interface

* *Interface Path*: /api/v1/bindings

* *Methods and Functions*:

  - POST: Trigger a cross-domain scheduling task.

  - GET /{bindingId}/status: Query the status of a scheduling task.

6.3. Data Plane API

6.3.1. Resource Reporting Interface

* *Interface Path*: /api/v1/metrics

* *Methods and Functions*:

  - PUT: Report real-time resource metrics for a domain.

```
{
  # Request example
  "cluster": "aws-us-east",
  "timestamp": "2023-07-20T08:30:00Z",
  "cpuUsage": 72.4,
  "memoryAvailable": 15.2
}
```

- POST /events：Push domain event notifications.

6.3.2. Status Synchronization Interface

* *Interface Path*: /api/v1/sync

* *Methods and Functions*:

  - PUT /configs: Synchronize domain configuration information.

  - POST /batch: Batch synchronization of status (supports up to 1000 objects per request).

6.4. Monitoring API

6.4.1. Real-time Query Interface

* *Interface Path*: /api/v1/monitor

* *Methods and Functions*:

  - GET /realtime: Retrieve a real-time resource monitoring view.

```
{
  # Response example
  "activeClusters": 8,
  "totalPods": 2450,
  "networkTraffic": "1.2Gbps"
}
```

6.4.2. Historical Data Interface

* *Interface Path*: /api/v1/history

* *Methods and Functions*:

  - GET /schedules: Query historical scheduling records.

    o startTime: Start time (in ISO8601 format).

- endTime: End time.
- maxEntries: Maximum number of entries to return (default is 100).

- GET /failures: Retrieve system failure history logs.

7. Scheduling Process

The DRO-DS scheduling process is designed to be modular and extensible, allowing for customization and adaptation to different environments. The process consists of four main steps:

7.1. Initializing Scheduling Tasks

1. *Task Request Reception*: The system receives a new scheduling task request, which includes information about the nature of the task, its priority, and resource requirements.
2. *Parameter Validation*: The system validates the task parameters to ensure their integrity and legality.
3. *Task Identification*: A unique identifier is assigned to the task to track and manage it throughout the scheduling process.

7.2. Collecting Domain Resource Information

1. *Total Resource Query*: The system queries each domain to determine the total available resources, including computing, storage, and network capacity.
2. *Single Node Maximum Resource Query*: The system checks the maximum available resources of each node within a domain to prevent resource fragmentation and scheduling failure.

7.3. Formulating Scheduling Policies

1. *Task Analysis*: The submitted task is mapped through the network modality mapping module to its resource requirements across three dimensions: storage tier, network identifier, and computing power (GPU performance).
2. *Domain Filtering*: Based on the task’s three-dimensional resource requirements and constraints specified in the application policy (such as region/cloud provider restrictions), domains with resource water levels exceeding limits are excluded, and unsuitable domains are filtered out.

3. *Candidate Domain Scoring*: The system evaluates the remaining candidate domains, considering factors such as current resource availability, task priority, resource constraints, and load balancing across domains. The weights of each factor are automatically adjusted based on real-time monitoring data, for example, increasing the network weight coefficient in scenarios with sudden traffic surges.

4. *Optimal Domain Selection*: The system selects the domain with the highest score to deploy the task.

7.4. Resource Allocation and Binding

1. *Task Allocation*: The system assigns the task to the selected domain to ensure its execution within the specified timeframe.

2. *Resource Status Update*: The system updates the resource status of the selected domain, recording the task assignment and resource utilization.

3. *Notification and Execution*: The system notifies the relevant modules of the task assignment and monitors its execution to ensure smooth operation.

8. IANA Considerations

This memo includes no request to IANA.

9. Security Considerations

* The DRO-DS system must operate within a trusted environment, such as a private cloud or enterprise-level infrastructure, where security measures are already in place. However, to ensure the security of the system, in accordance with the privacy considerations in Section 7.5 of [RFC7644], this standard requires the following measures when handling personally identifiable information:

  - *Authentication and Authorization*: All communications between components must be authenticated and authorized to prevent unauthorized access.

  - *Encryption*: Sensitive data, such as task configurations and resource metadata, must be encrypted both at rest and in transit.

- *Audit Logs*: Comprehensive audit logs must be maintained to track all system activities, facilitating troubleshooting and security investigations.

- *Isolation*: Resources and services must be isolated to prevent cross-domain attacks and ensure that failures in one domain do not affect other domains.

10. References

10.1. Normative References

[RFC7644] Hunt, P., Ed., Grizzle, K., Ansari, M., Wahlstroem, E., and C. Mortimore, "System for Cross-domain Identity Management: Protocol", RFC 7644, DOI 10.17487/RFC7644, September 2015, <https://www.rfc-editor.org/rfc/rfc7644>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/rfc/rfc5234>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

10.2. Informative References

[KarmadaDocs] "Karmada Documentation", n.d., <https://karmada.io/docs/>.

[KubernetesDocs] "Kubernetes Documentation", n.d., <https://kubernetes.io/docs/home/>.

Acknowledgements

This template uses extracts from templates written by Pekka Savola, Elwyn Davies and Henrik Levkowetz. [REPLACE]

Authors' Addresses

Chengyu Zhou (editor)
Huazhong University of Science and Technology
1037 Luoyu Road, Hongshan Distric
Wuhan
Hubei Province, 430074
China
Phone: 13375002396
Email: m202474228@hust.edu.cn

Yijun Mo
Huazhong University of Science and Technology
China
Email: moyj@hust.edu.cn

Hongyang Liu
Huazhong University of Science and Technology
China
Email: 3184035501@qq.com

Yunhui Pan
Huazhong University of Science and Technology
China
Email: panyunhui121@163.com