Nagare Media Engine: Towards an Open-Source Cloud- and Edge-Native NBMP Implementation

Matthias Neugebauer[1,2][a]

[1]*Department of Information Systems, University of Münster, Leonardo-Campus 3, 48149 Münster, Germany*

[2]*ZHLdigital, University of Münster, Fliednerstraße 21, 48149 Münster, Germany*

Keywords: Nbmp, Network-Distributed Multimedia Processing.

Abstract: Making efficient use of cloud and edge computing resources in multimedia workflows that span multiple providers poses a significant challenge. Recently, MPEG published ISO/IEC 23090-8 Network-Based Media Processing (NBMP), which defines APIs and data models for network-distributed multimedia workflows. This standardized way of describing workflows over various cloud providers, computing models and environments will benefit researchers and practitioners alike. A wide adoption of this standard would enable users to easily optimize the placement of tasks that are part of the multimedia workflow, potentially leading to an increase in the quality of experience (QoE). As a first step towards a modern open-source cloud- and edge-native NBMP implementation, we have developed the NBMP workflow manager `Nagare Media Engine` based on the Kubernetes platform. We describe its components in detail and discuss the advantages and challenges involved with our approach. We evaluate `Nagare Media Engine` in a test scenario and show its scalability.

1 INTRODUCTION

Multimedia processing is increasingly becoming more sophisticated. In order to increase the quality of experience (QoE), workflows now include advanced machine learning algorithms that optimize parameters based on the individual user session (Mueller et al., 2022). Furthermore, workflows are becoming more distributed as computing and caching are moved further to the edge of the network. This changing environment poses a challenge to workflow developers as they now have to integrate with different providers.

To meet these challenges, the Moving Picture Experts Group (MPEG) published ISO/IEC 23090-8 Network-Based Media Processing (NBMP) (ISO/IEC, 2020) which defines common APIs and data models. Workflows are broken up into multiple interconnected tasks, which are instances of functions that are deployed onto media processing entities (MPEs).

Ideally, an NBMP implementation can use the same MPE in different environments. We argue that Kubernetes is a good fit for that role, as it provides a common platform across cloud and edge providers. Beyond that, Kubernetes has emerged as a leading container orchestration system. Its scheduling mechanisms and built-in support for various workloads have matured in recent years. We, therefore, think that leveraging Kubernetes as part of an NBMP implementation would be beneficial. In this paper, we explore this idea by implementing an NBMP workflow manager that is based on Kubernetes and extends its functionality. In summary, the contributions of this paper are as follows:

1. A detailed description of how multiple Kubernetes clusters might be used as MPEs within an NBMP system.
2. `Nagare Media Engine` – An open-source NBMP workflow manager implementation that is based on the Kubernetes platform.

The rest of this paper is structured as follows. In the next section, we discuss NBMP as well as the Kubernetes platform as background for this paper. Section 3 gives an overview of the related work, after which we explain our implementation in Section 4. We discuss limitations in Section 5 and evaluate our approach in Section 6. Finally, Section 7 concludes this paper.

[a] https://orcid.org/0000-0002-1363-0373

Neugebauer, M.
Nagare Media Engine: Towards an Open-Source Cloud- and Edge-Native NBMP Implementation.
DOI: 10.5220/0012087200003538
In *Proceedings of the 18th International Conference on Software Technologies (ICSOFT 2023)*, pages 404-411
ISBN: 978-989-758-665-1; ISSN: 2184-2833

2 BACKGROUND

2.1 Network-Based Media Processing (NBMP)

NBMP describes a framework for network-distributed multimedia processing. It was standardized by MPEG as ISO/IEC 23090-8 in 2020 (ISO/IEC, 2020) with a second edition currently in development. NBMP defines a reference architecture split into a control and media plane, each containing several components as depicted in Figure 1.

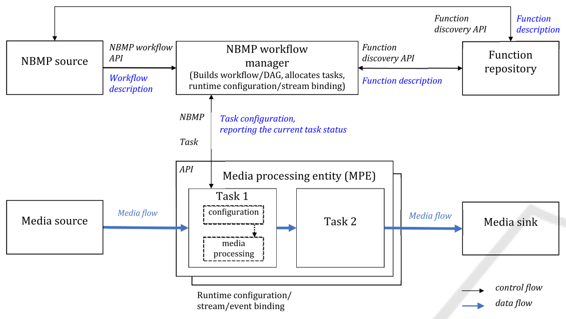

Figure 1: NBMP reference architecture (ISO/IEC, 2020).

The control plane consists of the following. New workflows are initiated by an *NBMP source*. It communicates with the *NBMP workflow manager* using the *Workflow API* to create, monitor, update and delete workflows. In an NBMP system, workflows form a directed acyclic graph (DAG) of media processing functions connected over a network. Available functions can be discovered via the *Function Discovery API* implemented by a *function repository*. The function selection can either be performed by the NBMP source or by the NBMP workflow manager based on provided constraints.

After the DAG is constructed logically, the NBMP workflow manager communicates with MPEs to instantiate the selected functions as *tasks*. How this communication takes place is outside the scope of the NBMP specification. This leaves room for several implementations in varying computing environments. Note that an NBMP workflow can span multiple MPEs. NBMP thus enables distributed multimedia processing for multi-cloud or edge scenarios. The workflow description allows specifying task scheduling constraints, but ultimately the NBMP workflow manager decides where a task is deployed to. The MPEs will signal the deployment status to the NBMP workflow manager. If all tasks are deployed successfully, the NBMP workflow manager will use the *Task API* for configuration and starting the execution.

Next to tasks, the media plane consists of *media sources* and *media sinks*. These represent the workflow inputs and outputs, respectively, and are part of the DAG. NBMP is suited for both streaming- and file-based scenarios. It is further agnostic to the streaming and media format. If a function is limited to certain formats, the function description needs to include these restrictions.

In total, NBMP defines three interfaces: the Workflow, Task and Function Discovery API. These should be implemented as Representational State Transfer (REST) APIs using JavaScript Object Notation (JSON) as an exchange format. The standard further specifies the JSON objects through JSON Schema (Wright et al., 2022) definitions. JSON objects that adhere to this schema are called workflow, task or function description documents (WDD, TDD, FDD), respectively. They share common descriptors also defined through JSON Schema. The MPEG published the schema definitions in a public repository[1] that already contains various improvements for the second edition. Our work is based on the updated schema.

NBMP describes the workflow and task lifecycles as state machines. The lifecycle starts in *instantiated*. After configuration, the state shifts to *idle*. It then transitions and stays in the *running* state as long as it receives a media stream. If an error occurs, the workflow or task is in the *error* state. Finally, the *destroyed* state signals the deletion. NBMP allows for interruptions in the media stream between tasks in case a task is reconfigured while being in the running state. In case of an error, multiple failover modes can be used.

2.2 Kubernetes

Kubernetes[2] is an orchestration platform mainly known for managing container workloads over multiple nodes. However, it is not limited to only orchestrating containers. It has thus become a "lingua franca" of cloud platforms. All major cloud providers have a managed Kubernetes offering that ties directly into the rest of the cloud platform, e.g. providing specific storage solutions for containers. Administrators can therefore use the same (or similar) API over multiple cloud providers. In recent years, Kubernetes has furthermore been used for edge computing. Where an upstream Kubernetes installation is not feasible, K3s[3] and Microk8s[4] provide a full Kubernetes distribution

[1]https://github.com/MPEGGroup/NBMP
[2]https://kubernetes.io/
[3]https://k3s.io/
[4]https://microk8s.io/

with a smaller footprint. KubeEdge[5] and SuperEdge[6] are solutions for devices with even stricter resource constraints and unstable networks between cloud and edge nodes.

At the core, the Kubernetes orchestration design evolves around resources and associated controllers. Resources provide a declarative way of specifying a certain state. For instance, the `Pod` resource describes a collection of tightly coupled containers that are deployed together on the same node. Resource types are part of a versioned group that forms a specific API. Resources are often represented in YAML Ain't Markup Language (YAML) and usually follow a specific structure. The `metadata` field is mandatory for all resources and at least specifies the name of the resource. Additionally, the namespace is required if the resource type is not scoped cluster-wide. Most resources further contain a `spec` field. It contains subfields that specify the desired state as defined by the user. The `status` field, on the other hand, contains subfields that describe the actually observed state. Kubernetes allows administrators to define custom resources through the `CustomResourceDefinition` (CRD) resource. As with built-in types, CRDs contain a versioned description of a type that is part of a specific group. JSON Schema is used to define the structure as well as validation rules. This feature allows third-party vendors to natively integrate with Kubernetes while users work with a familiar API. However, the Kubernetes API server is not responsible for the business logic associated with the resource. It only validates user input, possibly relying on webhooks for custom validation logic.

In order to reconcile the desired with the actual state, built-in and third-party controllers are running within the Kubernetes system. Controllers run a reconciliation loop that, for a given resource, continuously checks for a deviation from the desired state. In that case, steps are taken to reconcile the difference. Multiple controllers can work in unison to achieve a desired outcome. For instance, after the creation of a `Job`, the job controller will create a `Pod` according to the job specification. The scheduler, in turn, will update the `Pod` with a reference to a selected node. Finally, the node will spawn the actual containers and report the status in the appropriate fields of the `Pod`. The job controller will notice the status update and update the `Job` resource as well. This example also shows how different resources might relate to each other. `Pods` form the basis for higher-level constructs describing various workload types. This relationship can be expressed as an *owner reference* as part of the resource's `metadata`. In doing so, Kubernetes will automatically garbage-collect dependent resources once an owner gets deleted simplifying the controller cleanup logic.

As an optimization, controllers usually don't continuously check for changes but rely on notifications from the Kubernetes API server by defining a watch on a certain type. Furthermore, in-memory caching of resources plays a central role in efficient and scalable controller implementations. Third-party controllers can use the Kubebuilder framework[7] to take advantage of these optimizations.

3 RELATED WORK

NBMP is a fairly new standard only being published in 2020. Leading up to the publication, WIEN ET AL. provided an overview of the MPEG-I ISO/IEC 23090 standards collection NBMP is part of (Wien et al., 2019). XU ET AL. present NBMP in more detail in (Xu et al., 2022) and argue for its necessity. Various applications from the literature and the authors are discussed extensively. In the end, they encourage the development of NBMP implementations.

In (Ramos-Chavez et al., 2021), RAMOS-CHAVEZ ET AL. describe a testbed NBMP implementation that is used to evaluate various streaming functions for MPEG Dynamic Adaptive Streaming over HTTP (DASH) (ISO/IEC, 2019) and HTTP Live Streaming (HLS) (Pantos and May, 2017) streaming. The test results were then used for optimizations of the streaming solution.

YOU ET AL. implemented an NBMP prototype on the research stream processing platform World Wide Streams for solving computer vision problems (You et al., 2020, 2021).

The use of containers for NBMP functions has been proposed before (Bassbouss et al., 2021). The work of BASSBOUSS ET AL. involves using NBMP as a standardized workflow description for video-on-demand (VoD) media caching at the edge on trains. In a proof-of-concept, Docker containers provide various functions for transferring media to caches and monitoring the media streaming process.

Finally, MUELLER ET AL. describe an NBMP workflow that moves the transcoding to the edge to adapt the bitrate to the available bandwidth of the "last mile" (Mueller et al., 2022). This workflow additionally includes a microservice that uses machine learning for predicting optimal encoding parameters. AWS

[5]https://kubeedge.io/
[6]https://superedge.io/
[7]https://github.com/kubernetes-sigs/kubebuilder

Elastic Compute Cloud (EC2) is used to deploy a test setup. It is not detailed if and how the NBMP workflow manager was implemented.

As far as we know, an open-source NBMP implementation based on a modern, openly available platform such as Kubernetes is currently missing. Commercial off-the-shelf solutions do not yet exist either.

4 NAGARE MEDIA ENGINE

With `Nagare Media Engine`, we propose an NBMP implementation based on Kubernetes serving as MPE. This paper will only describe our NBMP workflow manager prototype. We take advantage of Kubebuilder in order to develop multiple efficient custom controllers. The codebase includes example configurations, is open source and can be retrieved from https://github.com/nagare-media/engine under the Apache 2.0 license. We intend to use `Nagare Media Engine` in a production scenario for live and VoD media but also think it can serve as a basis for future research. The following subsections will introduce the architecture and discuss each component.

4.1 Architecture Overview

In order to support multiple MPEs, `Nagare Media Engine` can talk to multiple Kubernetes clusters. One cluster fulfills the role of a management cluster that runs the NBMP control plane. All other clusters are used for running tasks and don't require any special components. We define multiple CRDs that are stored and reconciled on the management cluster (see Subsection 4.2).

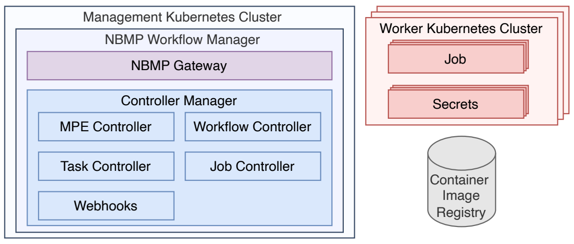

Figure 2: `Nagare Media Engine` architecture.

As depicted in Figure 2, our NBMP workflow manager consists of the *NBMP Gateway* and *Controller Manager*. The NBMP Gateway implements the Workflow API and translates WDD requests to and from our Kubernetes resources. The *Controller Manager* is a collection of Kubernetes controllers that observe our custom resources and try to reconcile the desired state. Additionally, it provides mutating and validating webhooks, i.e. HTTP endpoints called by the Kubernetes API server during the processing of our custom resources. The result of a successful reconciliation is the creation of one or multiple `Jobs` in worker Kubernetes clusters. The `Job` is a built-in resource type and is meant for workloads that run to completion.

We choose not to implement the Task API for the configuration and management of tasks. Instead, a `Secret`, another built-in type for securely storing and passing data, is mounted to the filesystem of the `Job`. This is closer to how other workloads are usually configured in Kubernetes. The main reason, however, is that we then no longer need to have network access to each deployed task. Even though Kubernetes has types to describe how incoming (HTTP) network requests should be routed, this can vary depending on the installed networking solution as well as additional components (network policies, service meshes, etc.). Our solution provides a more generic implementation that should work with many Kubernetes clusters. Although, for better compatibility with NBMP function vendors, we might need to develop a small translation layer and deploy it with each task. Functions must be packaged as container images and made available through a container image repository.

4.2 Custom Resource Definitions

We define six namespace-scoped custom Kubernetes resource types enabling multi-tenancy scenarios. Four additional types provide a cluster-scoped variant to allow administrators to define shared resources. These types have “`Cluster`” as the prefix in their names. Some types only store information and don't need to be reconciled.

The `Function` and `ClusterFunction` resources describe an NBMP function. The specification contains a `Job` template field that is used to instantiate `Job` resources. Furthermore, arbitrary default configuration values can be provided.

Access to media often requires authentication. The NBMP standard contains a security descriptor that can be associated with workflows, functions and tasks. Additionally, media is accessed through various transport protocols. To store this information, we defined the `MediaLocation` and `ClusterMediaLocation` resources. Authentication information is not stored directly in the custom resource. Instead, a reference to a `Secret` resource must be provided.

Users can define available MPEs through the `MediaProcessingEntity` and `ClusterMediaProcessingEntity` resources, respectively. MPEs may either be local, i.e. the worker Kubernetes cluster is identical to the management Kubernetes cluster, or remote. The status field will inform users whether a connection to the MPE has been established. Administrators can define cluster- or namespace-wide default MPEs through a special annotation in the resource metadata.

`Workflow` resources specify NBMP workflows. As workflows always run in a specific namespace, there exists no cluster-scoped version. The specification contains a list of references to `(Cluster)MediaLocations` as well as arbitrary configuration values. `Nagare Media Engine` controllers will update status fields to inform users about the execution state.

The `Task` resource specifies an NBMP task. `Tasks` contain references to a `Workflow`, `(Cluster) Function` and `(Cluster)MediaProcessingEntity`. Instead of a fixed reference to functions and MPEs, users may choose to use label selectors (i.e. key-value pairs). These are matched against labels defined in the metadata of the respective resources. The `Task` specification could also contain arbitrary configuration values that overwrite the defaults from the selected function. The status contains fields to inform users about the execution state of the `Task`.

As similar workflows are often executed on different inputs, we implemented the two optional resource types `TaskTemplate` and `ClusterTaskTemplate`. They contain the same fields as a `Task` except for the status. `Tasks` may choose to only reference a template. `Nagare Media Engine` will then merge the specifications during the reconciliation.

We implement Kubernetes API webhooks for all resource types in order to provide validation and default values. Our resource model splits an NBMP WDD into multiple smaller Kubernetes resources that reference each other. A referred resource might not be immediately available. Our implementation can handle this eventual consistency model which is typical for Kubernetes controllers.

4.3 NBMP Gateway

The NBMP Gateway implements the Workflow API and translates between our custom resources and WDDs. Each API request is validated according to the constraints defined in the NBMP standard. As our NBMP workflow manager implementation does not yet support all WDD descriptors, the validation also checks if the request can be fulfilled. The WDD `acknowledge` descriptor is added to all API responses and contains detailed information about invalid, partially fulfilled or unsupported items.

The NBMP specification allows NBMP sources to reference arbitrary NBMP function repositories. It is unclear to us how NBMP implementations that are built on different computing environments can share the same function repository. For instance, execution modes and packaging differ wildly for virtual machines, containers and serverless offerings. Additionally, allowing NBMP sources to reference arbitrary NBMP functions could pose a security risk. For these reasons, we choose to forbid requests that reference NBMP function repositories.

4.4 Media Processing Entity Controller

The MPE controller will reconcile `(Cluster)Media ProcessingEntity` resources. Its main purpose is to establish a connection to the configured Kubernetes cluster. A client object will be placed in an in-memory map that other controllers can use to manage resources in worker clusters. Additionally, it will instantiate a job controller (see Subsection 4.5) for each MPE. If the reconciliation is successful, the condition status field will indicate that the MPE is ready. On deletion, the connection is closed and the job controller stopped.

4.5 Job Controller

Each MPE will have an associated job controller instance. In contrast to the other controllers, it communicates with the API server of the worker Kubernetes cluster. The job controller only observes status changes of `Jobs` associated with NBMP tasks. We use metadata labels to differentiate these `Jobs`. Labels also specify the name and namespace of the `Task` in the management cluster since owner references only work within a single cluster.

The job controller serves as a link between the worker Kubernetes cluster and the task controller (see Subsection 4.7). Once a change to a `Job` is observed, it will trigger a reconciliation of the associated `Task` in the task controller.

4.6 Workflow Controller

The workflow controller reconciles `Workflow` resources by watching for changes to both `Workflow` and associated `Task` resources. This controller is structured into multiple phases roughly corresponding to the NBMP workflow lifecycle states (see Subsection 2.1). The current phase is reported to the user as a status field.

After the first reconciliation, the `Workflow` will be in the *initializing* phase. During each of the following iterations, the workflow controller tries to retrieve all `Tasks` belonging to this `Workflow`. Since we assume an eventual consistency model, the returned list might be empty. As soon as a `Task` is observed, the `Workflow` will be transitioned to the *running* phase. Because we also defined a watch on `Task` changes, the workflow controller will be informed about their current states. During each reconciliation, it counts the number of active, successful and failed `Tasks` and updates these numbers in the corresponding status fields. When all `Tasks` have been successful, we don't immediately transition to the *succeeded* phase. Instead, the `Workflow` will shift to the *awaiting completion* phase and is requeued for reconciliation after some time (by default 20 seconds). If no new `Tasks` are observed in the next iteration, it will transition to its final phase "succeeded". This intermediate phase helps to mitigate race conditions during the initial creation where very sort `Tasks` would bring the workflow to a termination state while new `Tasks` are still being added. This assumes all relevant resources are created within the duration of the waiting time. If the workflow controller observes a failed `Task`, it will either transition to the *failed* phase or ignore the error depending on the user-defined `Task` failure policy. Both "succeeded" and "failed" are terminal phases, i.e. new `Task` resources should not be created for this `Workflow` and will fail immediately if they are.

4.7 Task Controller

The task controller implements the main logic of our NBMP workflow manager. The reconciliation of `Task` resources is triggered in one of three ways: changes to the `Task` or the related `Workflow` or `Job` resources. The previously discussed job controller will inform the task controller about `Job` changes.

Similarly to the workflow controller, it is structured into multiple phases. Each iteration will check the status of the `Workflow` resource. If it is being deleted or has failed, the `Task` will be deleted or transitioned to the *failed* phase as well. The task controller will also make sure that an owner reference to the `Workflow` is set.

`Tasks` start in the *initializing* phase, where the task controller tries to determine the MPE and function according to the given definition. NBMP sources may specify a direct reference or only add labels that are used in the selection process. Alternatively, a `(Cluster)TaskTemplate` could be used. Finally, if not determined otherwise, the default MPE could be selected. The status is then updated with a reference to the selected MPE and function and the `Task` moves on to the *job pending* phase. Here the necessary information is compiled from the `Workflow`, `(Cluster)MediaLocations`, `(Cluster)Function`, `(Cluster)TaskTemplate` and the `Task` itself in order to create a `Secret` and `Job` in the worker Kubernetes cluster. An owner reference is placed on the `Secret` such that a deletion of the `Job` will clean up the `Secret` as well. After that, the `Task` is in the *running* phase until a termination of the `Job` is observed. Depending on the outcome, it will then transition to the *succeeded* or *failed* phase. For automatic retries in case of a failure, we instead rely on the mechanisms of the `Job` resource, i.e. if a `Job` fails, its error handling already failed as well.

4.8 Error Handling

Similarly to many other Kubernetes components, `Nagare Media Engine` is built on an eventual consistency model. As such, we expect failures to happen and instead, design the system to respond accordingly. The controller design around a reconciliation loop has led to robust systems such as Kubernetes itself. However, it can also pose challenges to the design of the controlled workloads, i.e. the function implementations in an NBMP system.

The ideal workload running in a Kubernetes cluster will tolerate interruptions. `Pods` may crash due to an error that occurred. However, the termination might also be "planned", i.e. deliberate. All `Pods` have an assigned priority. Higher priority `Pods` may preemptively displace lower ones in a resource contention situation. Administrators may also clear out Kubernetes nodes for maintenance or the Cluster Autoscaler could automatically remove the node entirely during a cluster scale-down.

Multiple strategies should be used by function vendors and Kubernetes administrators to handle this dynamic environment. At the very least, functions should not expect to be executed only once to handle restarts gracefully. Of course, a restart would still lead to a short interruption which might be unacceptable in the case of a live stream. In order to mitigate this, workloads in Kubernetes are usually replicated. Additionally, administrators can create `PodDisruptionBudget` resources to describe how many "planned" terminations a replicated workload might tolerate without downtime. Kubernetes will then delay further terminations until new replicas are running elsewhere. We think this can also be applied to the design of NBMP workflows. For instance, the DASH Industry Forum (DASH-IF) live media ingest protocol specification (DASH-IF, 2022) already describes an ingest protocol that can synchronize multiple incoming streams from redundant live encoders. Function authors should therefore not assume only one instance will run at the same time. In a more advanced mitigation strategy, functions might create checkpoints after some time or certain operations. However, we would generally suggest splitting long-running NBMP tasks into multiple smaller ones, e.g. splitting an encoding task into multiple chunks that are merged at the end. Next to a potentially higher parallelization, smaller tasks are less likely to be interrupted and provide natural checkpoints in an NBMP workflow system.

"Planned" terminations can also be an advantage for certain use cases. The `Pod` priority system enables mixed workloads on the same infrastructure. For instance, a new live stream workflow might postpone a low-priority VoD workflow that is resumed automatically after the live stream has ended. Additionally, a resilient function implementation has the advantage that Kubernetes administrators can utilize spot instances of cloud providers (i.e. virtual machines that are offered significantly cheaper but could be terminated preemptively at any time), potentially resulting in cost savings.

5 LIMITATIONS

Work on `Nagare Media Engine` is still ongoing as several NBMP descriptors are not yet supported. However, some descriptors also don't seem to apply to containerized environments (e.g. `static-image-info` subfields seem to refer to the provisioning of virtual machines). During the development process, we got the impression that the semantics of some NBMP descriptors is underspecified. Currently, a second edition of the NBMP standard is in development. We hope it will include a more detailed description of these aspects.

NBMP has a sophisticated way of specifying configuration parameters with constraints between values of different `Tasks`. These relationships are currently ignored.

As presented in this paper, `Nagare Media Engine` only implements the Workflow API. It would be beneficial if the NBMP Gateway would also provide the Function Discovery API such that NBMP sources can search for appropriate functions.

For our implementation, we choose Kubernetes `Secrets` for `Task` configuration in order to avoid granting network access to each `Task`. However, this implies that we do not support push-based inputs and pull-based outputs where the NBMP source is informed about network connection details of the `Task` that should receive the input or provide the output.

Finally, the observability of the workflow execution could be improved. We provide logging data and state information, but an integration into tracing and monitoring solutions is missing.

6 EVALUATION

To evaluate `Nagare Media Engine`, we created a management and worker Kubernetes cluster in Google Cloud. The worker cluster consisted of eight `e2-standard-2` nodes. We used the "autopilot" mode for the management cluster such that Google Cloud would manage the type and the number of nodes automatically.

Our main focus was to explore how the workflow manager would handle increasingly larger amounts of `Workflows`. In particular, we measured its CPU and memory usage as well as the total time to fully reconcile all `Workflows` of a given set. Each `Workflow` is identical and consists of two `Tasks` that simply sleep for 20 seconds. The `Job` template still contained CPU and RAM requests (0,4% of one CPU core and 12 MB RAM) such that the Kubernetes scheduler would deploy the `Jobs` evenly on the worker nodes.

In total, we tested four sets containing 50, 100, 500 and 1000 `Workflows`, respectively. For each set, the log output was gathered to determine the total execution time for a full reconciliation. These times thus also include the 20s run time of the `Jobs` as well as the time Kubernetes took for scheduling and instantiating `Pods`. Table 1 summarizes the measured absolute and relative times.

Table 1: Absolute and relative execution times for varying amounts of `Workflows`.

`Workflows`	Absolute time	Relative time
50	`1:21.614` m	`1.632` s/`Workflow`
100	`2:10.104` m	`1.301` s/`Workflow`
500	`8:53.503` m	`1.067` s/`Workflow`
1000	`16:48.991` m	`1.009` s/`Workflow`

While the relative times slightly improve with larger sets, the execution times still demonstrate an overhead involved with processing the `Workflows`. By default, reconciliation loops only process one resource at the same time. We rerun the experiment with four parallel reconciliations for the 1000 `Workflows` set but could not observe a significant difference (`16:44.797` m:s.ms). We suspect that the main contribution of the overhead comes from the actual scheduling and instantiation of the `Pods`.

Figure 3 depicts the CPU and memory usage for each set over time. While it increases with larger sets, it remains relatively low, leaving room for even larger

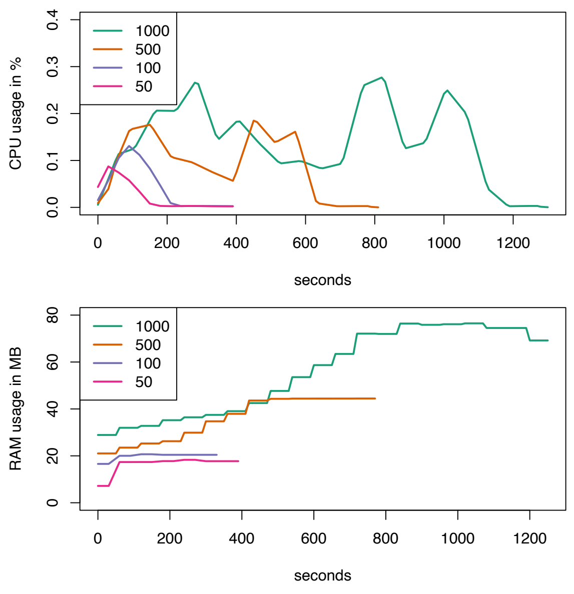

Figure 3: `Nagare Media Engine` CPU and memory usage.

installations. One can also observe that memory usage does not significantly decrease after all resources are reconciled. This is because the resources are still kept in the Kubebuilder cache.

7 CONCLUSION

In this paper, we discussed how we used Kubernetes as NBMP MPE in our workflow manager implementation `Nagare Media Engine`. Building upon the Kubernetes platform allows for a resilient NBMP system that can handle an eventual consistency model. However, this model might pose additional challenges for NBMP function developers. In an evaluation, we demonstrated how our workflow manager can scale to many concurrently executing workflows while being resource efficient at the same time.

Work on `Nagare Media Engine` is still ongoing. It is our plan to implement a library of compatible NBMP functions for efficient and reliable NBMP workflows. As the second edition of the NBMP standard is currently in work, we are eager to see how this technology will evolve in the future.

REFERENCES

Bassbouss, L., Fadhel, M. B., Pham, S., Chen, A., Steglich, S., Troudt, E., Emmelmann, M., Gutiérrez, J., Maletic, N., Grass, E., Schinkel, S., Wilson, A., Glaser, S., and Schlehuber, C. (2021). 5G-VICTORI: Optimizing Media Streaming in Mobile Environments Using mmWave, NBMP and 5G Edge Computing. In Maglogiannis, I., Macintyre, J., and Iliadis, L., editors, *Artificial Intelligence Applications and Innovations. AIAI 2021 IFIP WG 12.5 International Workshops*, volume 628, pages 31–38. Springer International Publishing, Cham. Series Title: IFIP Advances in Information and Communication Technology.

DASH-IF (2022). DASH-IF Live Media Ingest Protocol. Technical report, DASH Industry Forum.

ISO/IEC (2019). ISO/IEC 23009-1:2019 Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats. Standard, International Organization for Standardization, Geneva, CH.

ISO/IEC (2020). ISO/IEC 23090-8:2020 Information technology – Coded representation of immersive media – Part 8: Network based media processing. Standard, International Organization for Standardization, Geneva, CH.

Mueller, C., Bassbouss, L., Pham, S., Steglich, S., Wischnowsky, S., Pogrzeba, P., and Buchholz, T. (2022). Context-aware video encoding as a network-based media processing (NBMP) workflow. In *Proceedings of the 13th ACM Multimedia Systems Conference*, pages 293–298, Athlone Ireland. ACM.

Pantos, R. and May, W. (2017). HTTP Live Streaming. Technical Report RFC8216, RFC Editor.

Ramos-Chavez, R., Mekuria, R., Karagkioules, T., Griffioen, D., Wagenaar, A., and Ogle, M. (2021). MPEG NBMP testbed for evaluation of real-time distributed media processing workflows at scale. In *Proceedings of the 12th ACM Multimedia Systems Conference*, pages 173–185, Istanbul Turkey. ACM.

Wien, M., Boyce, J. M., Stockhammer, T., and Peng, W.-H. (2019). Standardization Status of Immersive Video Coding. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(1):5–17.

Wright, A., Andrews, H., Hutton, B., and Dennis, G. (2022). JSON Schema: A Media Type for Describing JSON Documents. Internet-Draft draft-bhutton-json-schema-01, Internet Engineering Task Force. Backup Publisher: Internet Engineering Task Force Num Pages: 78.

Xu, Y., Yin, J., Yang, Q., and Yang, L. (2022). Media Production Using Cloud and Edge Computing: Recent Progress and NBMP-Based Implementation. *IEEE Transactions on Broadcasting*, 68(2):545–558.

You, Y., Fasogbon, P., and Aksu, E. (2021). NBMP Standard Use Case: 3D Human Reconstruction Workflow. In *2021 International Conference on Visual Communications and Image Processing (VCIP)*, pages 1–1, Munich, Germany. IEEE.

You, Y., Hourunranta, A., and Aksu, E. B. (2020). OMAF4Cloud: Standards-Enabled 360° Video Creation as a Service. *SMPTE Motion Imaging Journal*, 129(9):18–23.