BIG HPC HIGH PERFORMANCE COMPUTING

A Management Framework for Consolidated Big Data and HP

Project number: 45924

Deliverable 1.2 - Platform architecture and APIs

Date	30 March 2021
Activity	1 - Platform Requirements, Design and APIs
Version	Final

Authors: João Paulo (INESC TEC); Ricardo Macedo (INESC TEC); Richard Todd Evans (UT TACC); Stephen Lien Harrell (UT TACC); Amit Ruhela (UT TACC); Bruno Antunes (Wavecom); Ricardo Leitão (Wavecom)

Reviewers: João Paulo (INESC TEC); Richard Todd Evans (UT TACC); Ricardo Leitão (Wavecom); Bruno Antunes (Wavecom), Vijay Chidambaram (UT)

Partners

INESCTEC LIP TEXAS The University of Texas at Austin MACC Minho Advanced Computing Center TACC wavecom

Funding

COMPETE 2020 Lisb@20 20 PORTUGAL 2020 UNIÃO EUROPEIA Fundo Europeu de Desenvolvimento Regional FCT Fundação para a Ciência e a Tecnologia UTAustin Portugal

Table of Content

Executive Summary

High-Performance Computing (HPC) infrastructures are increasingly used to support Big Data applications, whose workloads significantly differ from those of traditional parallel computing tasks. This is expected given the large pool of available computational resources, which can be leveraged to conduct a richer set of studies and analysis for areas such as healthcare, smart cities, natural sciences, among others. However, coping with the heterogeneous hardware of these large-scale infrastructures and the different HPC and Big Data application requirements raises new research and technological challenges. Namely, it becomes increasingly difficult to efficiently manage available computational and storage resources, to provide transparent application access to such resources, and to ensure performance isolation and fairness across the different workloads.

The BigHPC project aims at addressing these challenges with a novel management framework, for Big Data and parallel computing workloads, that can be seamlessly integrated with existing HPC infrastructures and software stacks. Namely, the project will develop novel monitoring, virtualization, and storage management components that can cope with the infrastructural scale and heterogeneity, as well as, the different workload requirements, while ensuring the best performance and resource usage for both applications and infrastructures.

These components will be integrated into a single software bundle that will be validated through real use-cases and a pilot deployed on both TACC and MACC data centers. Also, the proposed framework will be provided as a service for companies and institutions that wish to leverage their infrastructures for deploying Big Data and HPC applications.

This deliverable presents the architecture for the BigHPC framework while taking into account the requirements described at Deliverable 1.1. In more detail, the document describes the platform's generic design, interfaces and APIs, the internal organization of the main components (*i.e.*, Monitoring, Virtualization and Storage), as well as their integration details.

Glossary

API	Application Programming Interface
CLI	Command Line Interface
CPU	Central Processing Unit
CUDA	Compute Unified Device Architecture
GPU	Graphics Processing Unit
HPC	High-Performance Computing
I/O	Input/Output
INESC TEC	Institute for Systems and Computer Engineering, Technology and Science
LIP	Laboratory of Instrumentation and Experimental Particle Physics
MACC	Minho Advanced Computing Center
MPI	Message Passing Interface
QoS	Quality of Service
RAM	Random-Access Memory
RDMA	Remote Direct Memory Access
RPC	Remote Procedure Call
SDS	Software-Defined Storage
TACC	Texas Advanced Computing Center

1. Introduction

HPC infrastructures and services are no longer solely targeted at highly parallel modeling and simulation tasks. Indeed, the computational power offered by these systems is now being used to support advanced Big Data analytics for fields such as healthcare, agriculture, environmental sciences, smart cities, fraud detection, among others [OG+15, NCR+18]. By combining both types of computational paradigms, HPC infrastructures will be key for improving the lives of citizens, speeding up scientific breakthroughs in different fields (e.g., health, IoT, biology, chemistry, physics), and increasing the competitiveness of companies.

As the utility and usage of HPC infrastructures increases, more computational and storage power is required to efficiently handle the amount of targeted data-driven applications. In fact, many HPC centers are now aiming at exascale supercomputers supporting at least one exaFLOPs (10^{18} operations per second), which represents a thousandfold increase in processing power over the first petascale computer deployed in 2008 [RD+15, ECS+17]. Although this is a necessary requirement for handling the increasing complexity and scale of HPC applications, there are several outstanding challenges that still need to be tackled so that this extra computational power can be fully leveraged.

Management of heterogeneous infrastructures and workloads: By adding more compute and storage nodes one is increasing the complexity of the overall HPC distributed infrastructure and making it harder to monitor and manage. This complexity is increased due to the need of supporting highly diverse applications that translate into different workloads with specific data storage and processing needs [ECS+17].

Support for general-purpose analytics: The increased heterogeneity also demands that HPC infrastructures are now able to support general-purpose applications that were not designed explicitly to run on specialized HPC hardware and software environments, which was typically the case for traditional modeling and simulation applications [KWG+13].

Avoiding the storage bottleneck: As a complementary challenge, by only increasing the computational power and improving the management of HPC infrastructures it may still not be possible to fully harness the capabilities of these infrastructures. In fact, many applications are now data-driven and will require efficient data storage and retrieval (e.g., low latency

and/or high throughput) from HPC clusters. With an increasing number of applications and heterogeneous workloads, the storage systems supporting HPC may easily become a bottleneck [YDI+16, ECS+17]. As pointed out by several studies, the storage access time is one of the major bottlenecks limiting the efficiency of current and next-generation HPC infrastructures.

To sum up, the BigHPC project aims to address three main challenges: 1) improving the management of heterogeneous HPC infrastructures and workloads; 2) enabling the support for general-purpose analytical applications; and 3) solving the current storage access bottleneck of HPC services. Addressing these challenges is crucial for taking full advantage of the next generation of HPC infrastructures.

The goal of this deliverable is to further detail the design of the BigHPC framework while focusing on: i) the interfaces provided to HPC Users and System Administrators; ii) the internal architectures and APIs of the Monitoring, Virtualization and Storage components and; iii) the integration of these components into a fully functional framework.

The design and interface considerations discussed in this document are based on the functional and non-functional requirements collected at Deliverable 1.1 and on the requirements of the MACC and TACC supercomputers in terms of efficiently supporting a new generation of Big Data application.

The document is structured as follows: Section 2 discusses the general architecture for the BigHPC framework, the interfaces exported to Users and System Administrators, and overviews the integration of the main modules composing this framework. Then, Section 3 details the architectures and APIs for each of the main modules, namely the Monitoring, Virtualization and Storage components, while further highlighting their needs in terms of cross platform integration. Section 4 concludes this deliverable.

2. BigHPC Framework Architecture and Interfaces

BigHPC will design and implement a novel solution for monitoring and managing the infrastructure, data and applications of current and next-generation HPC data centers. The proposed solution aims at enabling both traditional HPC and Big Data applications to be deployed on top of heterogeneous HPC hardware. Also, it will ensure that resources (e.g., CPU, RAM, storage, network) are monitored and managed efficiently, thus leveraging the full capabilities of the infrastructure while ensuring that the performance and availability requirements of the different applications are met.

2.1. Architecture Overview

As depicted in Figure 1, The framework will support a deployment Command Line Interface (CLI) that will ease the adoption and use of the platform by Users to deploy their HPC and Big Data analytics applications as containerized jobs. Also, a management CLI interface will enable System Administrators to manage the overall infrastructure and deployed applications (e.g., to manage the lifecycle and placement of containers). Moreover, Users and System Administrators will have access to a monitoring web interface for checking the overall status of cluster resources and applications deployed on these (e.g., CPU, RAM, I/O usage at the compute node or job/container level). During this document we will provide further details on these interfaces.

The two CLI interfaces are provided by a logical component named Orchestrator that can include different management modules. In BigHPC, the Orchestrator will support two main modules, namely a Virtualization Manager and a Software-Defined Storage (SDS) Manager. However, by keeping this a logical and flexible component, the project aims at ensuring a straightforward integration of other management modules in the future (e.g., Software-Defined Networking Manager [KRV+15]).

The monitoring web interface will be achieved through a new Monitoring component that will collect resource usage metrics (e.g., CPU, RAM, network and storage I/O) for Big Data and parallel computing applications deployed at the HPC infrastructure. Short-term and long-term metrics will be collected at different granularities, namely at the job/container level and at the

compute node level. Furthermore, the Monitoring component will have access to general I/O metrics provided by the HPC storage backend (e.g., Lustre file system [Lustre+21]).

Besides exporting metrics through the web interface, the Monitoring component will provide an internal monitoring API so that the two Orchestrator managers can also have access to these metrics. This will be key for these modules to ensure a better management of HPC computational and storage resources.

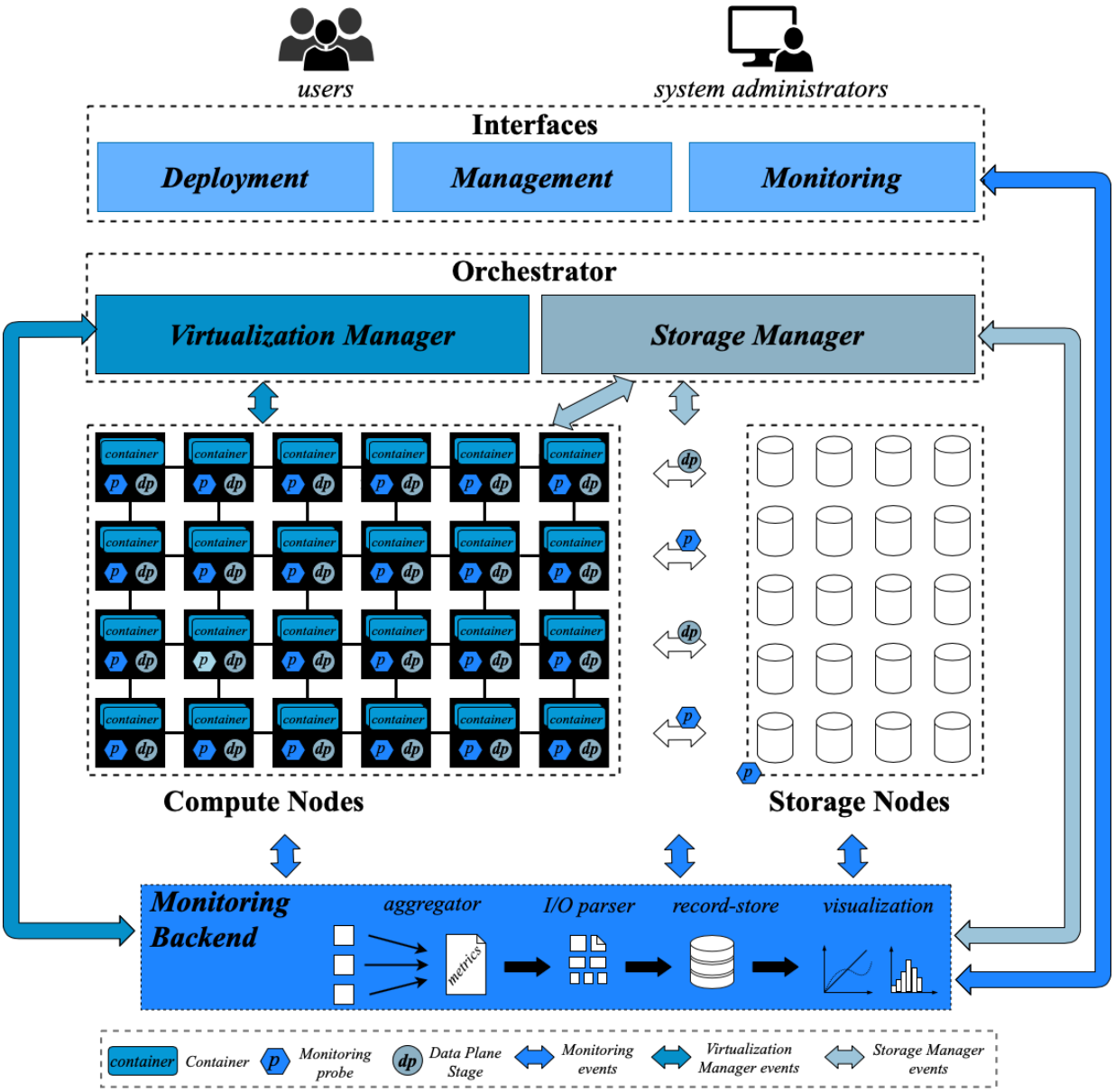

Figure 1: General BigHPC design.

The Virtualization Manager is responsible for abstracting the heterogeneous physical resources and low-level software packages (e.g., compilers, libraries, etc) that currently exist in an HPC cluster and to provide a common abstraction so that both HPC and Big Data applications can be easily deployed on such infrastructures. To achieve such a goal, this manager will resort to virtual containers. Containers will be running user applications in an isolated fashion and their placement, across the computational nodes, will take into account the performance and reliability requirements of each application while providing optimized usage of the infrastructures' overall resources (i.e., CPU, RAM, network, I/O, energy).

The Storage Manager will ensure optimized configuration of shared storage resources provided to applications and jobs running at the HPC infrastructure. This component will follow a Software-Defined Storage [MPP+20] approach while providing the building blocks for ensuring end-to-end control of storage resources in order to achieve QoS-provisioning, performance isolation, and fairness between HPC applications.

The Storage manager will decouple the control and data flows into two major components, control and data planes. The data plane is a programmable multi-stage component distributed along the I/O path that enforces fine-grained management and routing properties dynamically adaptable to the infrastructure status (e.g., rate limiting, I/O prioritization and fairness, bandwidth aggregation and flow customization). Such a component will be employed over containers and compute nodes at the HPC infrastructure and will manage the storage I/O flows to local storage mediums available at each compute node and the shared storage backend (e.g., Lustre file system). The control plane comprehends a logically centralized controller with system-wide visibility that orchestrates the overall storage infrastructure (i.e., data plane stages and storage resources) in a holistic fashion. The control plane is scalable and enforces end-to-end storage policies, tailored for attending the requirements of exascale computing infrastructures.

2.2. User Interfaces

We now further detail the interfaces exported to BigHPC Users and System administrators.

2.2.1 Deployment Interface

Container Scheduling and Management. Users will be able to specify workloads to the Virtualization Manager *Scheduler*. The specification will consist of a configuration file submitted through a CLI that prescribes the applications to run, how those applications interact, locations of input and output files, required resources, and resource affinities when appropriate. The Virtualization Manager will then schedule the workloads on the available resources, incorporating resource usage data from the Monitoring Component to ensure minimal interference of shared resources between workloads, and incorporating specifications from the Storage Manager to ensure resources are available to support a requested QoS. Users will also be able to query workload status, cancel their own workloads, and hold their own workloads through the CLI.

Before a workload is scheduled, the optimal container image to run the available resources will be obtained from the Virtualization Manager *Repository*. Users can download and upload container images and templates that support their workloads to the Repository through the CLI. They can also browse and search for available container images and templates through the CLI and delete any container images or templates that are owned by them. The Repository will match architectures to compatible or optimal container images and provide the Scheduler with the appropriate container images on request.

Storage quality of service. Users will be able to specify quality of service policies when deploying their applications/containers. The policies will be described through a simple configuration file (e.g., by using open standard formats such as JSON or YAML) that will be sent to the Storage Manager component. Also, these policies will be a subset of the management I/O storage policies, which we describe in more detail below. For example, Users may want to rate-limit the I/O bandwidth for a given application to ensure that it uses less resources and gets deployed faster (i.e., spends less time in the submission queue waiting for the needed resources).

2.2.2 Management Interface

Scheduling and Management. BigHPC Administrators will be able to start and stop BigHPC reservations - blocks of nodes requested through the site-native job scheduler - through the

CLI with an optional configuration file. Once the reservation is created the Virtual Manager Scheduler will be available to Users and Administrators. Administrators will be able to submit workloads, query workload status, cancel any workloads, and hold any workloads through the CLI. They may also modify resources or QoS assigned to any workloads before or after the workload is running.

Administrators will be able to access and modify the containers in the Virtual Manager Repository. They may modify the metadata describing the workloads and architectures the container supports. They can also modify the access permissions (e.g., read-only) to any container image.

Storage Resources Management. Again, System Administrators will be able to specify policies through simple configuration files (e.g., JSON or YAML) that will be submitted through a command line interface and sent to the Storage Manager component. This component will translate these instructions into storage I/O policies to be applied at the HPC infrastructure. In more detail, we aim at initially supporting the following policies.

- Specify a maximum I/O bandwidth (rate-limiting) for HPC applications.
- Ensure I/O bandwidth fairness across a set of HPC applications.
- Ensure transparent usage of both local and shared storage resources for HPC applications.

While the previous policies have the final goal of managing the storage resources being used by HPC applications, such can be achieved by applying them at distinct *granularities*, namely, for a given container (where the application is running), for the container(s) running on a specific set of compute nodes, for a set of containers belonging to the same User, for specific types of applications (e.g., AI, analytics, simulation), or even globally across the cluster.

BigHPC will explore these different granularity levels while also enabling a more fine-tuned control of what type of I/O requests to include in such policies. For example, it will be possible to distinguish data and metadata operations and specify different rate-limiting actions for each, which is an important aspect for ensuring a better usage of the shared file system backends at HPC centers.

Note that, as discussed previously, Users can also submit the above policies for their containers or applications through the deployment interface. In case there are any conflicting objectives, for example, the User is requesting more storage bandwidth for a container than the one specified by a System Administrator, the Administrator policies overrule conflicting rules from Users.

2.2.3 Monitoring Interface

The monitor interface will be responsible for supplying the collected metrics from compute nodes to both the Users, regarding their own jobs running on the cluster, and the Administrators providing insight on the cluster usage as a whole, while allowing them to have a real-time view of the system operation.

The *User Interface* displays user metrics regarding the containers running while allowing for real-time visualization of the status of the running jobs. Users will be able to query the monitor component, allowing them to profile the specific applications running at a given time and also compare it with past jobs previously submitted to the cluster.

The *Administrator Interface* will allow System Administrators to visualize the collected metrics by user or by node, or group of nodes, on a real-time basis and also access past cluster activity. It will also be responsible for relaying triggered alarms to the System Administrator while notifying them about a malfunctioning node or poorly behaved applications, for example, that are starving HPC resources.

2.3. Components Integration

Next, we describe how the three main BigHPC components, namely Monitoring, Virtualization and Storage Managers, interact with each other. Further details about the features, APIs and interfaces to do so are then provided, for each component, at Section 3.

As depicted in Figure 2, the Monitoring component must expose resource consumption metrics to the Virtualization and Storage Managers. As explained next, each of these managers may require different metrics (e.g., CPU, RAM, I/O) at different levels of granularity (e.g., per container, per compute node) for their operation. Therefore, the Monitoring component must enable an API with filtering or querying capabilities. Also, it will provide both

real-time metrics and long-term data as requested by the manager components. Note that these features are also important for the visual interface provided to Users and System Administrators.

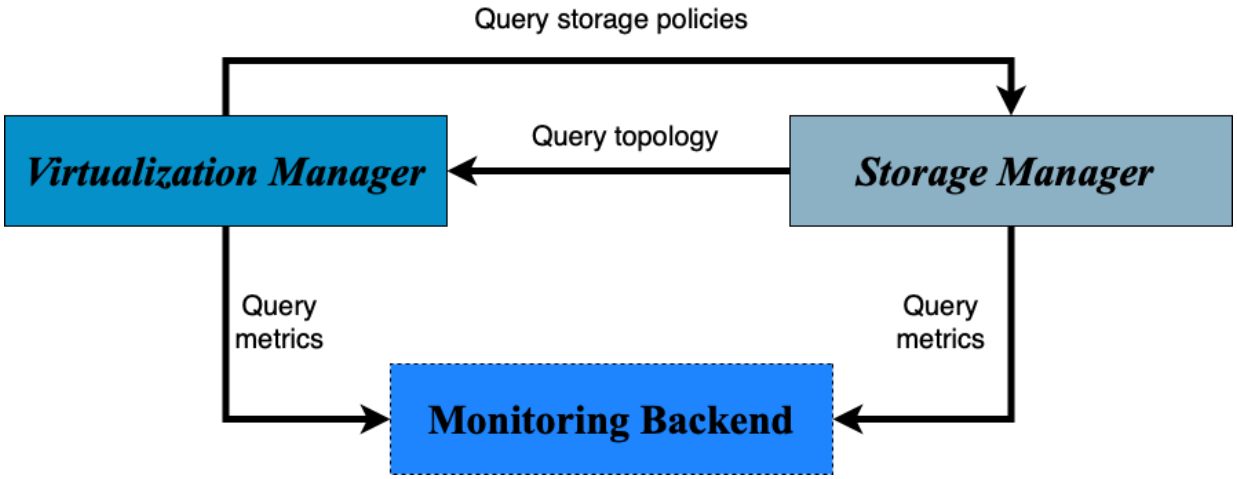

Figure 2: Dependencies between BigHPC components.

The Virtualization Manager will collect metrics such as: CPU and GPU utilization; local and central file system I/O; Memory consumption; and network interfaces usage, from the Monitoring component. Also, it will query the Storage Manager to gather information about in-place storage policies for specific Users, compute nodes or containers in order to find optimal placement strategies for new containers being launched. For instance, the Storage manager will have information about I/O rate-limits being imposed for containers deployed on a given compute node, therefore providing useful information about any leftover I/O storage bandwidth that may be used by another application being deployed there.

On the other hand, the Virtualization Manager will expose topology information about the placement of containers (i.e., the compute nodes where these are deployed on) and the users that launched them. This information will be queried by the Storage manager to build global cluster visibility and optimize storage policies across the infrastructure. Also, the Storage Manager will collect storage I/O metrics at the container, compute node and shared file system level from the Monitoring platform.

3. Components Architecture and APIs

This section describes the specific architecture and APIs for each of the BigHPC components.

3.1. Monitoring Component

The Monitoring Component is responsible for collecting, storing and providing the collected metrics from the system to both End-Users and System Administrators, for visualization and notification purposes. It will provide relevant monitoring metrics to the Virtualization Manager so it can choose the most suitable Compute Node(s) to deploy scheduled jobs (containers), and to the Storage Manager so it can monitor I/O storage metrics and take informed decisions. The monitoring component is in direct dependence of the Virtualization Manager, responsible for job scheduling, and it will request the setup of a monitoring probe on the compute node for each started job. These interactions are depicted in Figure 3. As explained previously, the Orchestrator is a logical component that includes both the Virtualization and Storage Managers.

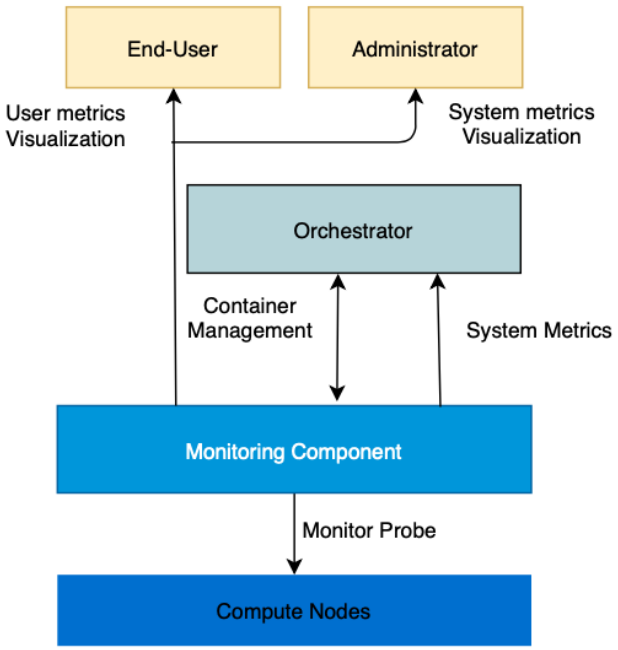

Figure 3: Components interaction diagram for the Monitoring Component.

3.1.1 Internal Monitoring Backend

The Monitoring Backend will support two different interfaces: an Internal interface that is responsible for the communications inside the Monitoring Component; and an External one that allows other components to interact with the Monitoring Backend.

As shown in Figure 4, the Monitoring Backend will serve as a middleware between the Users, System Administrators, Orchestrator and the remaining monitoring components, namely the Collectors and the Metrics Database. The Monitoring Backend will interact with the internal components by means of the following actions:

- Send Metrics
- Get Metrics
- Store Metrics

Send Metrics. The Monitoring Agents, or Collectors, will be responsible for gathering the specified metrics in each Compute Node and for each job deployed by the Virtualization Manager. This granularity will ensure that even short-term jobs will get profiled by the Monitoring Component. The data gathered by local collectors will be shared by a message broker to be stored in the Metrics Database on a periodic basis that can be configured by System Administrators.

Get Metrics. To ensure the best usage of the available Compute Nodes, the Virtualization and Storage Managers may poll for a specific set of metrics from the Compute Nodes directly from the Collectors in real time and also at setup time before job scheduling .

Store Metrics. The Metrics Database will be responsible for storing the metrics in a time series database for further processing and Visualization by Users and System Administrators. This long term storage will enable a better understanding of past jobs that ran at the HPC infrastructure.

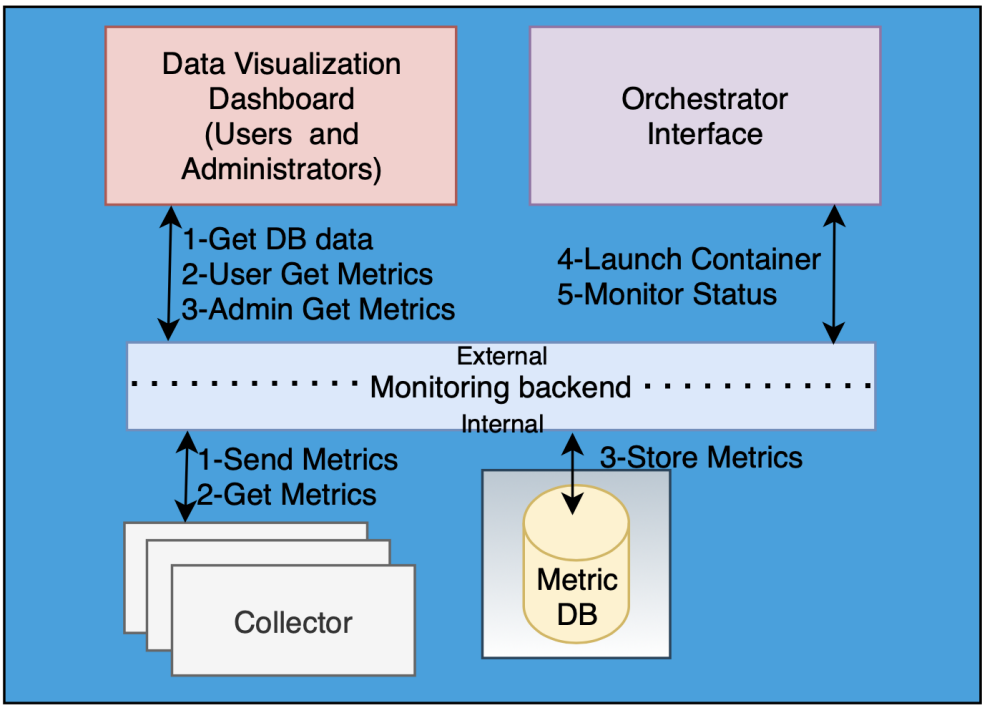

Figure 4: Architecture of the Monitoring Component.

3.1.2 External Monitoring Backend

It is through the External Monitoring Backend that Users and System Administrators will have access to the Data Visualization Dashboard. This Dashboard provides an interface for viewing stored metrics and also for gathering real time information about running jobs of each User. The Visualization Dashboard will also trigger alarms for misbehaved or starving jobs and display the current Compute Nodes availability. Additional through this Dashboard the Users and System Administrators may also request the following actions:

- Get DB Data
- User Get Metrics
- Admin Get Metrics

Get DB Data. For a more detailed analysis the System Administrators may directly query the Metrics Database and/or export it to other systems to perform offline evaluation of the

deployed jobs at the HPC infrastructure. This may also be used to backup the Database to an external source.

User Get Metrics. Send the metrics to the Users Data Visualization Dashboard upon user's request. The users will be able to query their entries in the database to visualize it in the dashboard. They may also download this information and store in their systems for further analysis.

Admin Get Metrics. Send the metrics to the System Administrators Data Visualization Dashboard. This will allow Administrators to have access to current and past performance metrics from the HPC infrastructure in their Visualization Dashboard.

The Orchestrator Interface is made available to the Virtualization and Storage Managers, so that these can interact with the Monitoring Backend. The requests to the monitoring interface will be done by a REST API, and will allow the following operations:

- Launch Container
- Monitor Status

Launch Container. This operation will allow the Virtualization Manager to send information to the Monitoring Backend regarding the scheduling of a container, to correlate it with the corresponding user, and to enable the monitoring collector for this job to be deployed on the corresponding Compute Node(s).

Monitor Status. This operation will be used by the Orchestrator components to collect metrics about deployed jobs and the overall HPC infrastructure. In more detail, the Virtualization Manager will be able to gather information about the utilization of computational resources at different Compute Nodes (e.g., RAM, CPU, GPU), which will enable the placement of new jobs in the most suitable nodes. The Storage Manager will have access to I/O monitoring metrics for deployed containers and for the local and shared storage backends (e.g., compute node local disks and Lustre file system) to ensure that storage policies are being enforced correctly and adjust them if needed.

3.2. Virtualization Manager

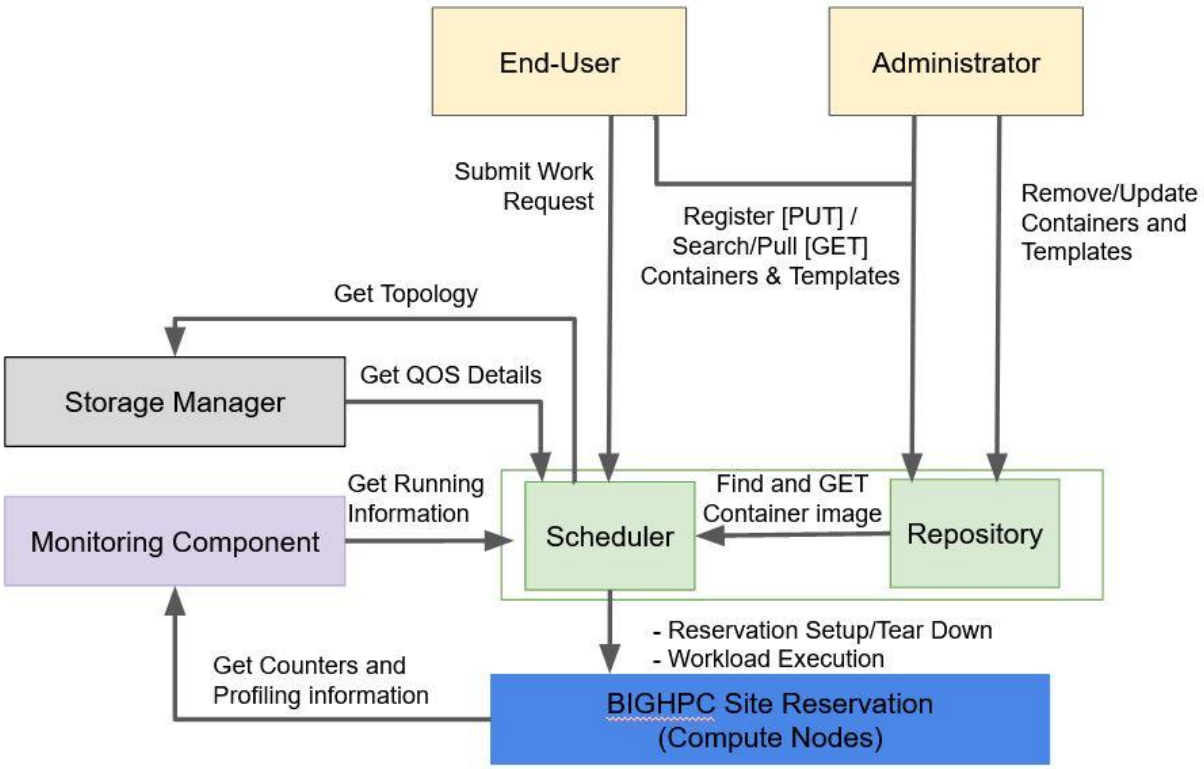

Figure 5: Component Interaction diagram of Virtualization Manager.

As depicted in Figure 5, the Virtualization Manager (VM) is composed of two services - the BigHPC *Scheduler* and the BigHPC *Repository*. The Scheduler interfaces with the HPC site's native job scheduler (e.g. SLURM) and is responsible for processing, scheduling, and running user submitted BigHPC workload requests. The Scheduler optimally schedules workloads and associated QoSs based on information it obtains by querying the BigHPC Monitoring Component through the Monitoring API and Storage Manager through the Deployment API, respectively.

The Repository stores container images and build-templates and provides container discovery and provisioning. It is the means by which users submit and access the containerized software necessary to support their workload requests. Both services are accessible through an API exposed as a Command Line Interface (CLI).

The Scheduler and Repository may be hosted on external servers so long as they have access to a site native job submission node (e.g. login node).

3.2.1 Virtualization Manager Scheduler

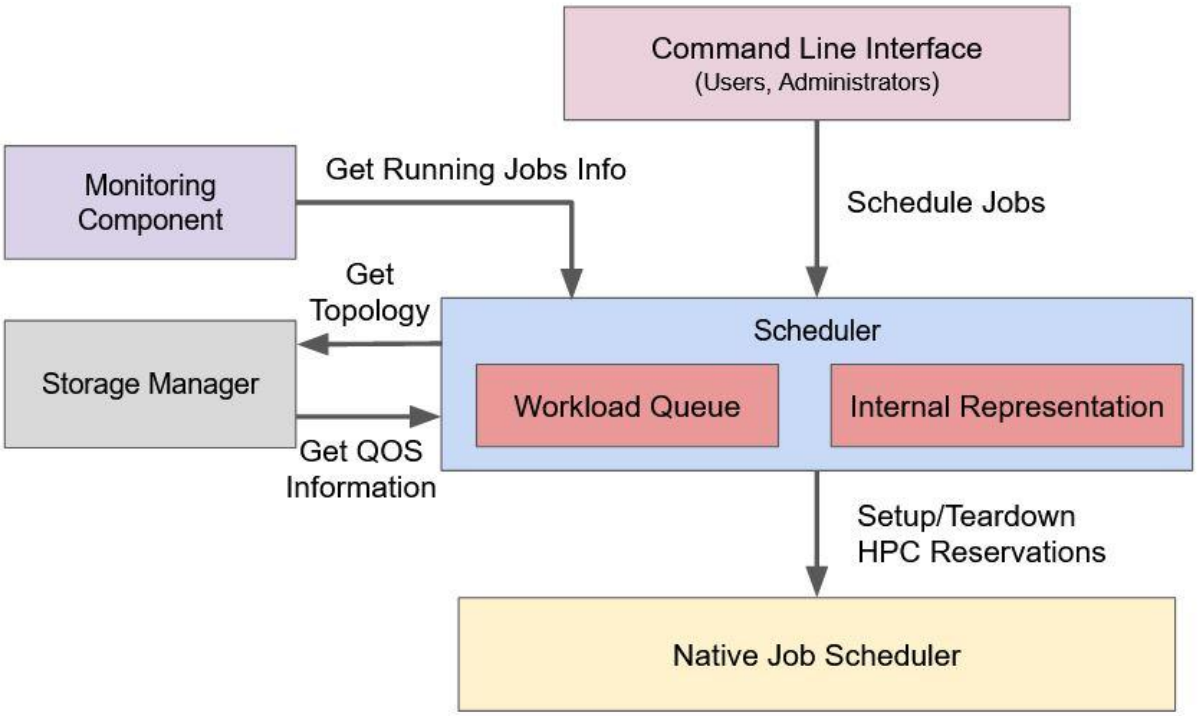

Figure 6: Architecture of BigHPC Scheduler.

The Scheduler manages BigHPC workloads (Figure 6). Users submit workload requests to the Scheduler in the form of JSON or YAML files. These workload requests specify the system, resources, and software necessary to execute the request. The Scheduler stores these requests in a queue until the *BigHPC Reservation* is set up on the requested system by a BigHPC Administrator, and the Reservation has the available resources to execute the request. The setup of the BigHPC Reservation first launches a job within the site's native job scheduler (e.g. SLURM), which effectively reserves resources (computes nodes, cores, GPUs, etc.), then starts a daemon that builds and maintains a representation of executing BigHPC workloads. This representation includes the locations those workloads are running within the BigHPC Reservation (affinity to compute nodes, cores, GPUs, etc.). The same daemon accepts user workload requests, queries the Repository for the correct container images, queries the BigHPC Monitoring component for resource usage, queries the Storage Manager for QoSs,

and runs those requests such that they interfere minimally with currently executing workloads. Users and System Administrators will submit requests to the Scheduler via a CLI that allows them to perform the following actions:

- BigHPC Reservation Setup & Tear Down
- Workload Submission [POST]
- Workload Query/Hold or Modify/Cancel [GET/PUT/DELETE]

BigHPC Reservation Setup & Teardown. A major capability that the BigHPC infrastructure enables is the ability to schedule workloads on resources in a way that minimizes interference and provides specified Quality of Service within a traditional HPC environment. HPC sites typically have their own native job schedulers which manage the execution of user jobs but no notion of potential sources of resource contention or interference between jobs. BigHPC interfaces with a site's native job scheduler by building and submitting a native job script processed from a BigHPC Administrator's Reservation request submitted in a JSON or YAML file. The Reservation request specifies the system and resources to be reserved for the BigHPC Reservation. There is also a command to tear down the BigHPC Reservation.

The Scheduler maintains an internal representation of the workloads and their affinities to the resources of the Big HPC Reservation. It also maintains a queue of workloads that can not yet be run due to unavailable resources. A global QoS can be assigned to the BigHPC Reservation which will be determined by querying the Storage Manager through the Deployment API.

Workload Submission [POST]. Users submit workload requests to the Scheduler via a CLI and a JSON or YAML workload specification file. The workload specification file describes the hardware resources, software resources, and optionally the QoS requested. It also describes the workload to be run, whether services are persistent throughout the lifetime of the workload, and the location of input and output.

The hardware resource description specifies device counts and affinities for each component of the workload. The software resource description specifies the necessary container image to obtain from the Repository for each component of the workload. The Scheduler will copy the necessary container images from the Repository before the workload is scheduled to run. The QoS specifies the network bandwidth and IOP rate the workload requests and is determined by matching requested QoSs to their definitions in the Storage Manager. Individual QoSs are

superseded by the BigHPC QoS. If the Scheduler is unable to satisfy a request it will notify the user as soon as possible.

Workload Query/Hold or Modify/Cancel [GET/PUT/DELETE]. Users can query the status of their active workload requests: Queued, Running, or Held. They can cancel or hold a request, deleting it from the queue or preventing it from running until it is unheld. They can also modify the hardware resources or QoS for any active request. These requests are submitted via a CLI to the Scheduler.

3.2.2 Virtualization Manager Repository

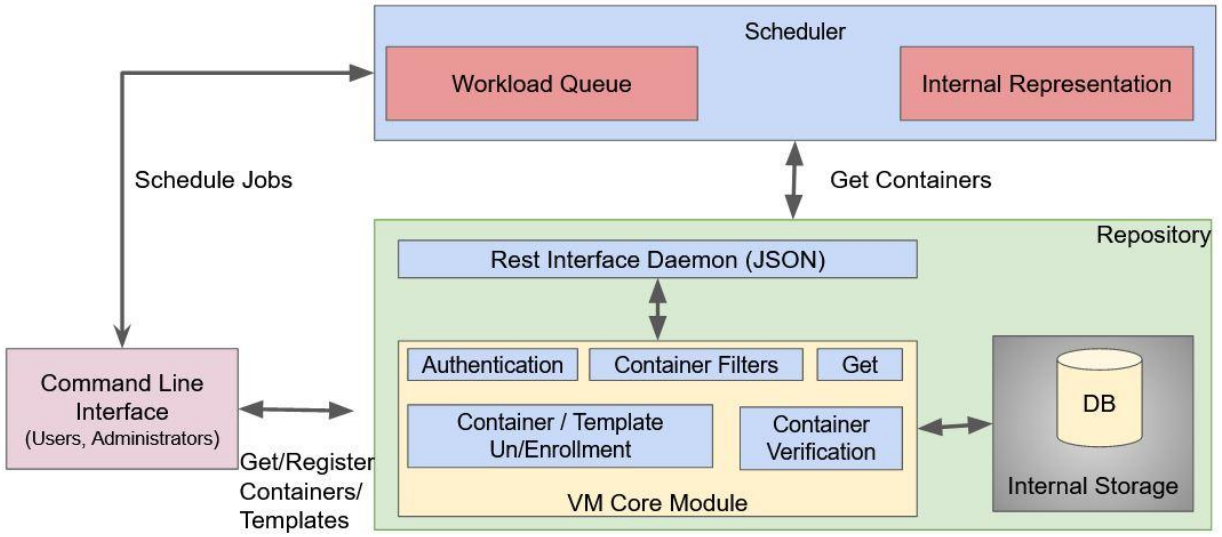

Figure 7: Architecture of BigHPC Repository.

As depicted in Figure 7, the Repository will be used to access and maintain the container images used to support requested workloads. The container images should support a particular system/hardware configuration and will typically have user installed applications. The Repository will consist of a database, a module specifying an API to the database, and a daemon that uses that module API to service requests from Users (e.g. Users, Scheduler, Administrators). Users will be able to submit commands via a CLI to the daemon which allows them to make the following requests:

- *Container Discovery [GET]*

- *Container Image and Template Provisioning [GET]*
- *Container Image and Template Registration [POST/DELETE]*

The container storage will be implemented as an Object Store database such as MongoDB. The Repository may be external to the Scheduler which manages the workloads so long as tcp and ssh access is available. Docker templates used to build available containers will also be stored in the Repository if available. Descriptions of each request type that interacts with the Repository are below.

Container Discovery [GET]. Users may explore what containers (and associated Docker templates) are registered in the Repository. There will be commands submitted to the daemon through a CLI to perform this exploration. These commands will check the Repository and return a manifest of what containers are available. The manifest will include each container and metadata such as which user registered the container, what hardware it supports, what applications it supports, what services it supports, and what container runtime it requires. These same metadata will be used to label each container image. User requests may filter return results by container characteristics such as CPU and GPU architecture, network, and application including compilers and dependencies. Service level metadata including # containers/templates registered with and disk usage will be queryable. Containers will also be labeled by HPC system compatibility when appropriate.

Container Image and Template Provisioning [GET]. Users may request registered container images and docker templates via the CLI. The commands will return the container description and copy the image or template to a location specified by the user in the command. The user may then take that container to a local machine and modify it as necessary and then register the modified container with the Repository. They may also request the docker templates used to construct any of the registered containers and subsequently use the templates to build whatever container images they require.

Container Image and Template Registration [POST/DELETE]. Registering a container image makes it available to the BigHPC Scheduler and other Users for discovery and GET requests. This process requires:

1. Submitting the container specification to the Repository via the CLI. The location of the container image or docker template to submit must be specified in the command. The

User must input required metadata including; which user registered the container, what hardware it supports, the name of the application/workload it supports, what services it supports, and what container runtime it requires. These metadata may be specified via command line arguments or a YAML file.

2. The Repository daemon will copy the container or docker template used to build it to the Repository server and store the container in the "container pool" (Object Store Database). The container pool/database configuration will be specified as part of the Repository configuration file.
3. The Repository daemon will run a validation command that checks the container specification against discoverable container characteristics and also verifies the container does not already exist. The container is successfully registered with the Repository upon completion and available for use. If it cannot be verified it will try to provide feedback to the User as to why (missing fields, incorrect specification).

Finally, container images may also be deregistered (deleted) using the CLI.

3.3. Storage Manager

HPC storage resources will be managed by following a design based on the SDS principles, namely through a control plane and data plane as depicted in Fig 8. The Control plane is a logically centralized component, although physically distributed for scalability and fault-tolerance purposes. As explained in Section 2.2, it receives QoS policies (e.g., I/O bandwidth, scheduling, fairness) from Users and System Administrators and enforces these holistically throughout the data plane.

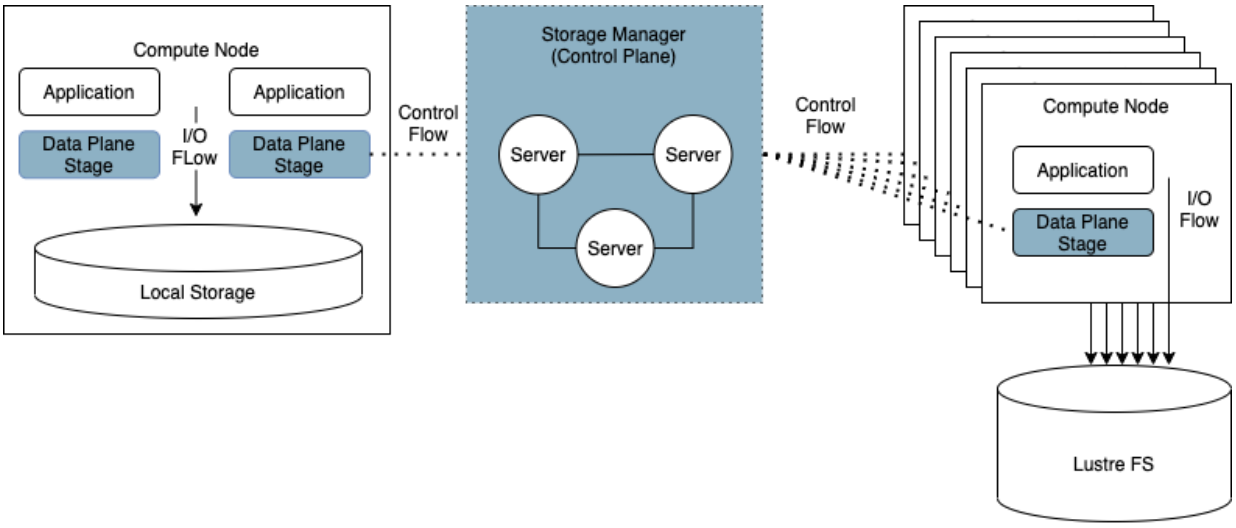

Figure 8 - Architecture Overview for the Storage component.

The Data plane is composed of a set of stages that sit between HPC applications/containers and the corresponding storage mediums. In BigHPC, the storage mediums can be local storage drives at the compute nodes or the shared file system deployed at these infrastructures (e.g., the Lustre file system). The goal of these data plane stages is to enforce rules sent by the Control Plane (e.g., rate-limiting of I/O requests). Next, we further detail the architecture and interfaces for the control and data plane components.

3.3.1 Control Plane

The control plane, also referred to as Storage Manager in this document, is composed by the components depicted in Figure 9.

The Metadata Store module keeps the user-defined policies being enforced at the HPC infrastructure. These will be stored in a database that maps different policies (e.g., rate-limiting) to the corresponding data plane stages where they are going to be applied to.

Moreover, this component caches partial information about the monitoring metrics and topology of the overall HPC infrastructure on two other databases. The monitoring information will hold relevant metrics about data plane stages and the overall HPC infrastructure such as CPU, RAM, network and storage I/O (including both local disks and shared file system I/O metrics).

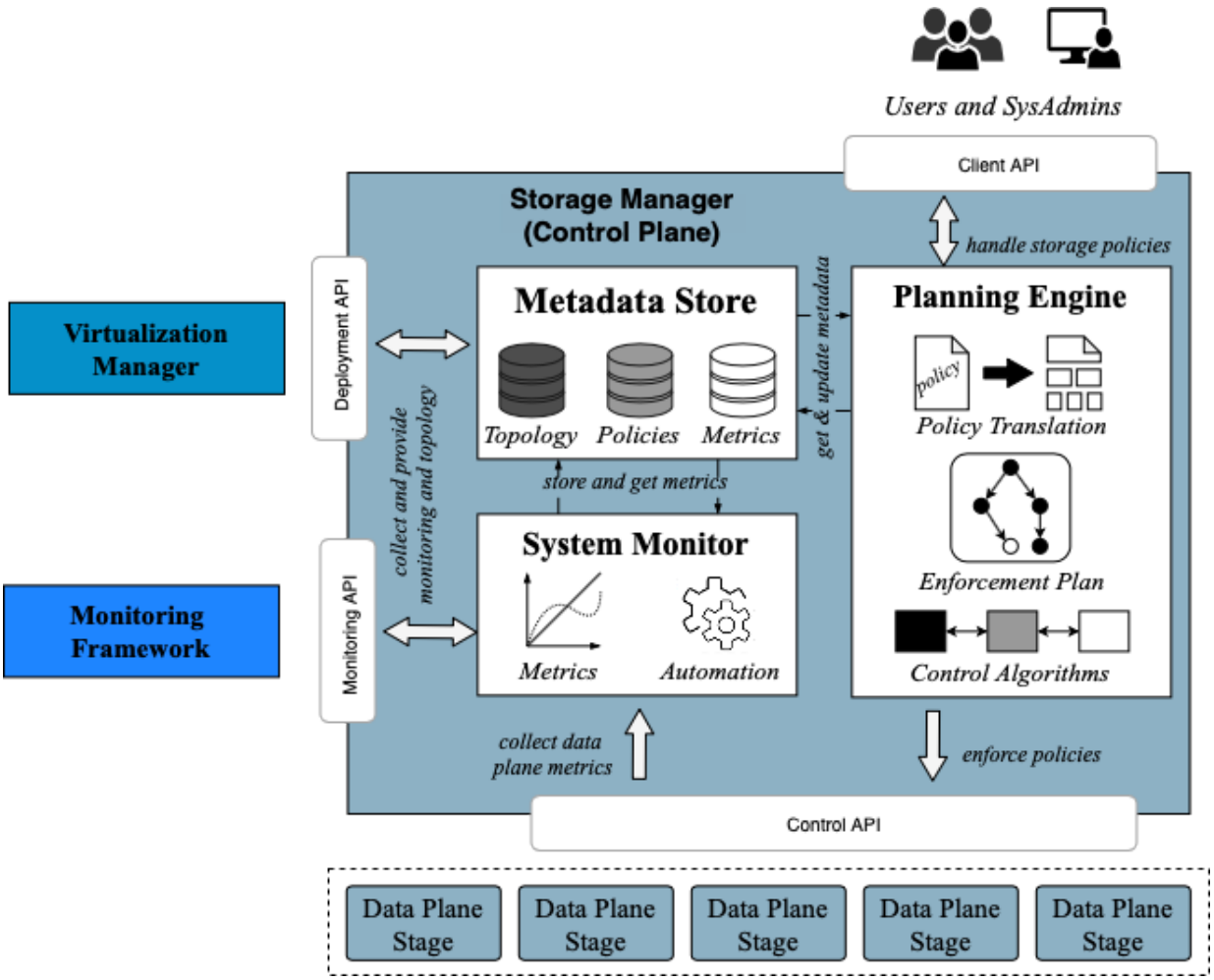

Figure 9 - Storage Manager Architecture.

The topology information (e.g., graph describing the distribution of storage and compute nodes, containers and data plane stages) will enable gathering additional information about a given data plane stage such as the compute node where it is deployed, the type of storage medium being accessed by it, or the container and User id associated with that data plane stage.

Topology data is collected through a Deployment API that will enable the Metadata Store to remotely query the Virtualization Manager, which is responsible for holistically managing the deployment of applications and containers at the HPC infrastructure, in order to fill its local cache. On the other hand, the Deployment API will be used by the Virtualization Manager to query the policies being enforced at the Storage Manager (control plane). The latter

information is valuable to do an optimized deployment of containers according to their storage requirements and the available resources at the HPC infrastructure.

The System Monitor module is responsible for collecting, aggregating, and transforming unstructured metrics and statistics from data plane stages (e.g., IOPs), as well as, system-level metrics from the BigHPC Monitoring framework (e.g., storage resources' utilization, compute nodes' CPU, GPU and RAM usage). After being pre-processed, these metrics are stored at the metrics database of the Metadata Store module. Information will be collected through two distinct APIs. The Monitoring API will be used to query the BigHPC Monitoring framework, while the Remote Procedure Call (RPC) Control API will be used to query information from each deployed data plane stage.

The monitoring information, along with metadata concerning the user-defined policies and the infrastructure topology, will be queried, combined and used by the Planning Engine. At this engine, user-defined policies are parsed, validated, and translated into stage-specific rules that can be holistically enforced for a single data plane stage (e.g., limit IOPs rate for a specific container) or a number of stages to perform distributed enforcement (e.g., I/O fairness or prioritization across containers of a given User).

The policy enforcement strategy is calculated through different control algorithms that specify how the data plane handles I/O flows and define the most suitable place for policies to be enforced. Examples of such control algorithms will include proportional sharing, isolation and priority, feedback control, and machine learning techniques. The actions to be taken at each data plane stage will leverage RPC control API.

The Storage Manager will be deployed across multiple dedicated servers in order to have a scalable and fault-tolerant design. Namely, by leveraging the computational power of multiple servers it will be possible to have a control plane that can handle the large volume of monitoring metrics and topology information being collected, while taking timely and accurate decisions, specified by the user-defined policies, across hundreds to thousands of control plane stages. A distributed design will also enable tolerating failures of control plane nodes without affecting the operation of the HPC infrastructure. Note that this will require the use of coordination protocols across control plane nodes to ensure a coherent and holistic vision of HPC resources.

3.3.2 Data Plane

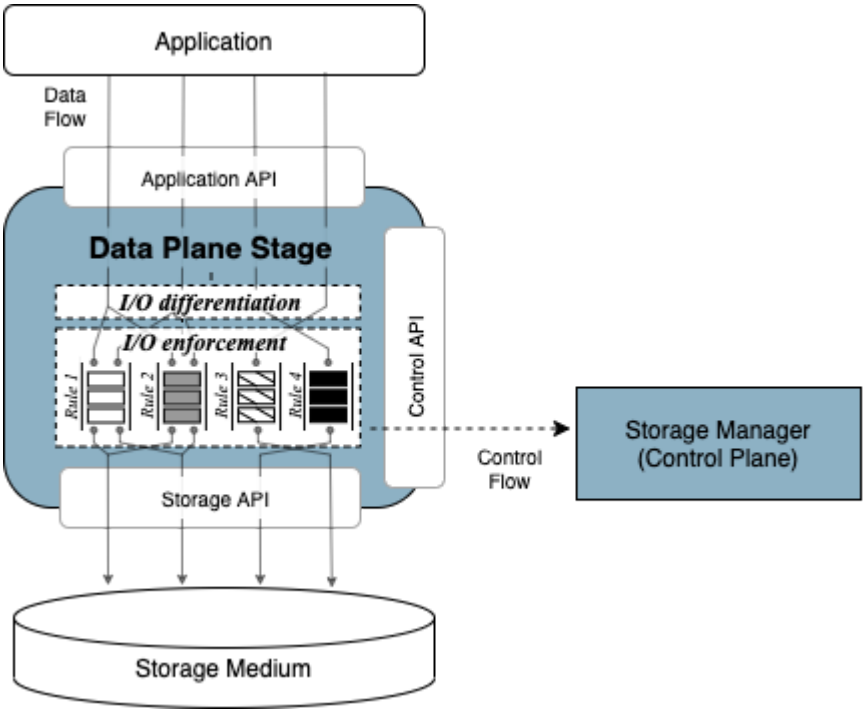

Figure 10 - Data Plane Architecture.

As explained previously and depicted in Figure 10, the data plane component will be composed of multiple independent stages that sit between applications or containers and storage resources (e.g., local storage medium or shared distributed file system). The goal is to have a distinct data plane stage per application / container, while the control plane is responsible for coordinating multiple stages when user-defined policies must be applied in a group-wise fashion.

Each stage must be able to intercept I/O requests (Application API) from containers or applications. This will be done by following two approaches:

- An intrusive approach that requires changing the code of applications to intercept and differentiate I/O calls (e.g., metadata vs data, write vs read, background vs foreground operations) while providing more control on the set of requests and corresponding arguments that can be collected at the cost of less transparency and portability;
- A non-intrusive approach that provides standard storage interfaces, such as POSIX or block device APIs, which can be used by unmodified applications. Namely, this can be

achieved by resorting to tools such as FUSE or LD_PRELOAD [FUSE+21, LD+21]. This approach provides greater portability and transparency for applications at the cost of losing some of the differentiation capabilities (e.g., differentiating background and foreground tasks) provided by intrusive techniques.

After collecting and differentiating I/O requests, each stage will be responsible for enforcing rules (e.g., rate-limiting, encryption, compression) over specific operations of the applications (e.g., data, metadata, background, foreground). The actions that must be applied are defined and communicated, through the RPC Control API, by the control plane, which has the necessary information and holistic visibility to do so. As explained previously, the Control API will also be used by each stage to send monitoring metrics to the control plane. Finally, storage requests will be forwarded by the data plane stage through a Storage API that will be compatible with the corresponding storage medium (e.g., file system or block device).

In BigHPC, we will provide a library to build custom data plane stages. The main goal is to have a generic solution that enables a simpler and faster implementation of stages, while promoting code-reuse and portability for a wide range of applications and container technologies.

4. Conclusion

This deliverable details the architecture and interfaces of the BigHPC platform. As discussed throughout the document the proposed design will enable HPC Users and System Administrators to gain access to new monitoring, management and deployment interfaces. These will be key to achieve a better control of HPC resources and the traditional HPC jobs and Big Data applications deployed across them. Also, the design is aligned with the requirements of state-of-the-art infrastructures, such as the ones supported by TACC and MACC.

As another contribution, the deliverable discusses the internal architectures and interfaces for the BigHPC's Monitoring, Virtualization and Storage components. Also, it shows how these components interact and integrate with each other.

The information contained in this document will drive the next major step of the project, namely, the implementation of the BigHPC framework.

References

[ECS+17] Joseph, E., Conway, S., Sorensen, B., Thorp, M., 2017. Trends in the Worldwide HPC Market (Hyperion Presentation). HPC User Forum at HLRS.

[FUSE+21] FUSE (File system in Userspace). https://github.com/libfuse/libfuse. Accessed in 2021.

[KRV+15] Kreutz, D., Ramos, F.M., Verissimo, P., Rothenberg, C.E., Azodolmolky, S. and Uhlig, S., 2015. Software-defined networking: A comprehensive survey. Proceedings of the IEEE, 103(1), pp.14-76.

[KWG+13] Katal, A., Wazid, M. and Goudar, R.H., 2013, August. Big data: issues, challenges, tools and good practices. In 2013 Sixth international conference on contemporary computing (IC3) (pp. 404-409). IEEE.

[LD+21] LD_PRELOAD man page. https://man7.org/linux/man-pages/man8/ld.so.8.html. Accessed in 2021.

[Lustre+21] Lustre web page. https://www.lustre.org. Accessed in 2021.

[MPP+20] Macedo, R. Paulo, J. Pereira, J. and Bessani, A. 2020. A Survey and Classification of Software-Defined Storage Systems. ACM Comput. Surv. 53, 3, Article 48 (June 2020), 38 pages.

[NCR+18] Netto, M.A., Calheiros, R.N., Rodrigues, E.R., Cunha, R.L. and Buyya, R., 2018. HPC cloud for scientific and business applications: Taxonomy, vision, and research challenges. ACM Computing Surveys (CSUR), 51(1), p.8.

[OG+15] Osseyran, A. and Giles, M. eds., 2015. Industrial applications of high-performance computing: best global practices (Vol. 25). CRC Press.

[RD+15] Reed, D.A. and Dongarra, J., 2015. Exascale computing and big data. Communications of the ACM, 58(7), pp.56-68.

[YDI+16] Yildiz, O., Dorier, M., Ibrahim, S., Ross, R. and Antoniu, G., 2016, May. On the root causes of cross-application I/O interference in HPC storage systems. In 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS) (pp. 750-759). IEEE.