

Bachelor's Thesis

submitted in partial fulfillment of the
requirements for the course "Applied Computer Science"

Interactive Data Center Digital Twin using Virtual Reality

Lars Quentin

MatrNr: 21774184

First Supervisor: Prof. Dr. Julian Kunkel

Second Supervisor: Dr. Sven Bingert

Georg-August-Universität Göttingen


Institute of Computer Science


ISSN 1612-6793

December 15, 2022


Georg-August-Universität Göttingen
Institute of Computer Science

Goldschmidtstraße 7
37077 Göttingen
Germany

 +49 (551) 39-172000

 +49 (551) 39-14403

 office@informatik.uni-goettingen.de

 www.informatik.uni-goettingen.de

Abstract

Virtual Reality (VR) allows users to experience and interact with digital environments in an immersive way and, therewith, can provide a more intuitive way of interacting with computers using more natural human-machine interaction paradigms. Additionally, virtual environments can provide various advantages for data visualization, as the digital space is not constrained by real-world physical limitations. Furthermore, leveraging the third dimension and the spatial features, VR can be a viable alternative to traditional visualization.

The goal of this thesis is the interaction and visualization paradigms to the realm of data centers by creating a server room as a digital twin using the Unity game engine. By integrating the live utilization metrics of a real High-Performance Computing (HPC) cluster into the virtual world, the VR server room can be used to analyze the current data and reason about its implications. This integration is done by incorporating the data into several places in the virtual environment. Following the creation of this digital twin, the application was evaluated against traditional, web-based data dashboards by conducting a user acceptance study. The results imply that the VR alternative improved the user experience and reduced the analysis complexity compared to the web version, concluding that VR can be a valuable addition to traditional metric visualization for the application of data center monitoring.

Acknowledgements

First and foremost, I would like to thank my supervisors, Prof. Dr. Julian Kunkel and Dr. Sven Bingert, for their valuable guidance and support throughout this thesis. The feedback received shaped the direction and outcomes of my work.

Furthermore, I am also grateful to my parents for their support throughout my life, as well as to Dr. Ulrich Degenhardt for introducing me to the joys of programming, Linux, and L^AT_EX.

Contents

List of Figures	vi
List of Abbreviations	viii
1 Introduction	1
1.1 Motivation	1
1.2 Goals	2
1.3 Contributions	2
1.4 Structure	4
2 Background	5
2.1 Virtual Reality	5
2.1.1 Devices	6
2.1.2 Game Engines	10
2.1.3 Interaction Guidelines and Cybersickness	12
2.2 HLRN	13
2.3 Usage and Monitoring of HPC Systems	14
3 Related Work	16
3.1 Dashboards and Visualisation	16
3.2 VR Data Visualisation	19
3.3 Data Center VR	20
4 Methodology and Design	25
4.1 VR Technology Stack	25
4.1.1 VR Headset	25
4.1.2 Game Engine	26
4.2 VR Environment and Interaction	29
4.3 2D Plotting in VR	34
4.3.1 Evaluation of Existing Libraries	34
4.3.2 Implementation Approaches	35
4.4 Live Metrics	36
4.4.1 Metrics Used	37
4.5 User Acceptance (UA) Study	39
5 Implementation	42
5.1 Unity and Virtual Environment	42
5.2 Plotting Library	43
5.3 Live Metrics	45
5.3.1 Query Design	45
5.3.2 Singleton Structure	47
5.3.3 Fetching and Parsing Metrics	49
5.3.4 Event Subscribers	51
5.3.5 Offline Usage	55
5.4 UA Study	55
5.4.1 Data Generation	55

5.4.2	Test Cases	57
6	Evaluation	61
6.1	Participants and Study Structure	61
6.2	Analysis Methodology	61
6.3	Results	62
6.4	Discussion	64
7	Conclusion	65
	References	66
A	Grafana Proxy Queries	75
A.1	Central Processing Unit (CPU) Data Query	75
A.2	CPU Data Example Response (Truncated)	76
A.3	Memory Data Query	77
A.4	Memory Data Example Response (Truncated)	78
A.5	Graphics Processing Unit (GPU) Data Query	79
A.6	GPU Data Example Response (Truncated)	80
A.7	Storage Data Query	81
A.8	Storage Data Example Response (Truncated)	82
B	UA Study Template	83
B.1	Part 1. Topic Explanation	83
B.1.1	How the Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen (GWDG) HPC System works	83
B.1.2	How to submit a job	83
B.1.3	VR Environment	84
B.2	Part 2. Pre-Questionnaire	90
B.3	Part 3. Test Cases	91
B.3.1	Test Case 1: Are the servers used optimally?	91
B.3.2	Test Case 2: How does the scheduler work?	91
B.3.3	Test Case 3: Long Term Decision Making	91
B.4	Part 4. Post Questionnaire	91
C	Mocked Data used in UA Study	93
C.1	Test Case 1: Are the servers used optimally?	93
C.2	Test Case 2: How does the scheduler work?	96
C.3	Test Case 3: Long Term Decision Making (Data Set 1)	98
C.4	Test Case 3: Long Term Decision Making (Data Set 2)	101

List of Figures

1	Compute nodes of HLRN-IV in Göttingen [4].	1
2	Popular Grafana Dashboard for Monitoring Kubernetes with Ingress-NGINX and Prometheus [7].	2
3	The CAVE setup at the UIC [15].	5
4	An assembled Google Cardboard VR mount [31].	6
5	A Samsung Gear VR, smartphone mounted [34].	7
6	A Quest 2 VR [39].	8
7	Use cases for the Quest 2: (a) for collaborative remote work, (b) for general interaction, (c, d) for gaming, and (e) for drawing and sculpting 3D visuals.	9
8	A Valve Index [51].	10
9	The metrics server structure at the GWDG	15
10	For bar charts, the y-axis should start at zero.	16
11	The x-axis should have a consistent interval size.	17
12	The most important information should be highlighted in order to most efficiently communicate the data.	17
13	Example of the color blindness simulator Coblis [87].	18
14	An example of a bad data-ink ratio. This graph contains 5 points of data [89].	19
15	Tufte’s redesign of the box-plot, maximizing the data-ink ratio [89].	19
16	Leaseweb 360-degree data center tour on YouTube [93].	20
17	IronMountain VA-1 360-degree data center tour on YouTube [94].	21
18	Manually controlled 360-degree tour through one of Hetzners data center [95].	21
19	A recording of a 6-DoF-based recording of Green Mountains data center [98].	22
20	A VR solution for configuring a server rack [100].	22
21	The various VR native interactions provided by the simulator [101].	23
22	The interface of the VR construction software used internally by Meta [21].	23
23	The interface and streamed data supported by the VMWare VR Datacenter Experience [103].	24
24	The PC Building Simulator [105].	24
25	Comparison matrix for different VR hardware	26
26	A top view of the Virtual Environment (VE) layout	29
27	The overview in the starting view and the possible interaction provided.	30
28	The button mapping of the Oculus Quest 2 controller used.	31
29	The CPU and GPU server rooms	32
30	The lamps of a CPU and GPU node and their related metrics	32
31	The tabular data overview, interacted with by scrolling using the right ray.	33
32	The server User Interface (UI) of a GPU rack.	34
33	All plot types supported by Plotty	35
34	GPU based 2D rendering done by the Open Source frames per second (FPS) counter Graphy [128]	36
35	The Scientific Compute Cluster (SCC) cluster [135]	38
36	All server models in Blender.	43
37	The UML diagram of Plotty’s line and bar chart classes	44
38	The triangle mesh of the line between m_{i-1} and m_i	44

39	An example of a Query solely built with Grafana’s InfluxQL query builder	46
40	Viewing the Queries used by Grafana with the Firefox Developer Tools . .	47
41	The UML class diagram for the GrafanaSingleton	48
42	A comparison of Unity JavaScript Object Notation (JSON) parsers [146]. .	51
43	All event subscribers related to the UIs and server lights	52
44	All event subscribers related to the initial overview	53
45	The “Frontends and Storage” dashboard	58
46	The “Overview + All Nodes” dashboard	59
47	The “Single Node Dashboard” dashboard	60
48	Distributions of the the self-reported post-questionnaire data about the VR user experience.	62
49	Distributions of the the self-reported post-questionnaire data about the VR analysis.	63
50	The distribution of the self-reported data answering the question “How did the VR version change the quality of the experience?”.	64
51	Test Case 1: Mocked Frontend Data	93
52	Test Case 1: Mocked Storage Data	93
53	Test Case 1: Mocked CPU Node Data	94
54	Test Case 1: Mocked GPU Node Data	95
55	Test Case 2: Mocked Frontend Data	96
56	Test Case 2: Mocked Storage Data	96
57	Test Case 2: Mocked CPU Node Data	97
58	Test Case 3.1: Mocked Frontend Data	98
59	Test Case 3.1: Mocked Storage Data	98
60	Test Case 3.1: Mocked CPU Node Data	99
61	Test Case 3.1: Mocked GPU Node Data	100
62	Test Case 3.2: Mocked Frontend Data	101
63	Test Case 3.2: Mocked Storage Data	101
64	Test Case 3.2: Mocked CPU Node Data	102
65	Test Case 3.2: Mocked GPU Node Data	103

List of Abbreviations

AI Artificial Intelligence

AIO All-In-One

API Application Programming Interface

AR Augmented Reality

ARM Advanced RISC Machines

BI Business Intelligence

BIM Building Information Modeling

CAVE CAVE Automatic Virtual Environment

CLI Command Line Interface

CPU Central Processing Unit

DoF Degrees of Freedom

DX Developer Experience

EULA End User License Agreement

FIFO First-In-First-Out

FLOPS Floating Point Operations Per Second

FOV Field of View

FPS frames per second

GOAL Game Oriented Assembly Lisp

GPU Graphics Processing Unit

GWDG Gesellschaft für wissenschaftliche Datenverarbeitung mbH Göttingen

HDRP High Definition Render Pipeline

HLRN Norddeutsche Verbund für Hoch- und Höchstleistungsrechnen

HMD Head-Mounted Display

HPC High-Performance Computing

IDC International Data Corporation

IMU Inertial Measurement Unit

JSON JavaScript Object Notation

MDC Modular Data Center

MIT Massachusetts Institute of Technology

MOOC Massive Open Online Course

NDK Native Development Kit

OOP Object-Oriented-Programming

OS Operating System

PoC Proof of Concept

RAM Random-Access Memory

REST Representational State Transfer

SCC Scientific Compute Cluster

SDK Software Development Kit

STL Standard Template Library

TSDB Time Series Database

UA User Acceptance

UAT User Acceptance Test

UI User Interface

UIC The University of Illinois at Chicago

URP Universal Render Pipeline

USB Universal Serial Bus

UX User Experience

VE Virtual Environment

VFX Visual Effects

VPN Virtual Private Network

VR Virtual Reality

XR Extended reality

1 Introduction

The first section discusses the motivations, goals, and contributions of this thesis. Section 1.1 outlines the need for big data methods in data-driven research as well as the complexities for researchers and other end users of understanding HPC environments, which could be solved by VR. Continuing in Section 1.2, the objectives of the thesis get defined. Derived from this, Section 1.3 describes the technical contributions of the work. Finally, in Section 1.4, the structure of the thesis is explained.

1.1 Motivation

According to the International Data Corporation (IDC), worldwide data volume increased from 2 zettabytes in 2010 to 64.2 zettabytes in 2020 with a five-year compound annual growth rate of 23 percent [1]. These trends, together with the recent advances in big data tools and methodology also result in more data-intensive research discovery methods in many traditionally not computing-reliant sciences. This sparks the discussion of data exploration being the fourth science paradigm besides empirical, theoretical, and computational sciences [2] [3]. Due to this, many researchers from a variety of fields use many computational and data-driven methods for their research. To process these vast amounts of data, many researchers rely on HPC infrastructures for their computations.



Figure 1: Compute nodes of HLRN-IV in Göttingen [4].

Understanding the server topology and their current utilization can be complicated, especially for researchers and other HPC users unfamiliar with scientific computing infrastructures. Although very verbose and detailed, technical documentation can be perceived as overwhelming for a researcher unfamiliar with typical HPC workflows. While centralized monitoring systems like Grafana [5] for on-premise or Datadog [6] for cloud-based monitoring do exist, they are not suitable for end users for multiple reasons: Firstly, they can expose many security or privacy-related pieces of information. Secondly, they are primarily designed with system administrators in mind, resulting in feature-rich but complex UIs in the form of dashboards, which often require a deep knowledge of the data center structure and server configuration. Furthermore, dashboards do not preserve spatial or structural inter-node information, such as their relative physical location. Lastly, due to security reasons, it also is not possible to make the server rooms physically accessible to

all end users.



Figure 2: Popular Grafana Dashboard for Monitoring Kubernetes with Ingress-NGINX and Prometheus [7].

Another technology to aid the problem of visualizing and educating complex topics is Virtual Reality. VR has gained much popularity over the last years, resulting in a global revenue increase of 1.8 billion USD in 2016 to 3.3 billion USD in 2019 [8]. With the advancements of All-In-One (AIO), Android-based VR headsets popularized by the Oculus Quest series¹ with its resulting decrease in cost also became more accessible to developers, resulting in better tooling and frameworks.

VR technologies can improve the current utilization visualization and general HPC education. It can leverage immersive technologies to communicate a high-level overview of a HPC configuration and its usage.

1.2 Goals

The goal of this thesis is to create an immersive, VR-native visualization tool for data center metrics. To make the association between the servers and their metrics as intuitive as possible, a digital twin of a server room has been created as a VE. This VE allows the visualization and interactive analysis of real metrics from actively used HPC systems. Furthermore, the live streaming of data metrics was integrated.

1.3 Contributions

To fulfill the above goals, a Unity-based [9] server room digital twin with the following features has been implemented:

¹Later renamed to Meta Quest

- The program is fully controllable by VR native interaction methods. No classical input methods such as keyboards are required, thus the immersion is never interrupted.
- For each server, there is a 2D UI located in the 3D world space. There the metrics are rendered through the use of plots, similar to those found in standard dashboard technologies.
- Furthermore, multiple overviews are integrated into the Virtual Environment.
- The data can be live-streamed from an InfluxDB [10] Time Series Database (TSDB) proxied through Grafana.
- The application is designed to be used entirely offline, which is especially convenient for conferences and other presentations. This is possible through the use of mocked, artificially generated data that is shipped as part of the application.
- All assets, plugins, or other parts used for this application are open source and compatible with the MIT license [11], allowing the free distribution of the complete source code, enabling any Oculus Quest 2 owner to build and use the program.

During the implementation of these objectives, some additional technical requirements became apparent. Thus, the following other contributions had to be created within the scope of the thesis:

- In order to facilitate the 2D plotting while retaining the right to publish the complete source code, a standalone Unity plotting library was developed. This library, also licensed as MIT [11], supports line plots, bar charts, and gauge plots.
- For the real-time metrics streaming of Grafana dashboards, based on InfluxDB databases, a new event-driven, asynchronous, and thread-safe polling architecture was designed and implemented. Additionally, it supports the inclusion of precomputed mock data for offline usage.
- A workflow and implementation for procedurally generating mock data was developed. Additionally, a python based client was created to either generate C# classes containing that data or stream the data to an InfluxDB using the InfluxQL-based Application Programming Interface (API).

Lastly, as part of this thesis, the following non-technical contributions have been made:

- An comprehensive analysis of the current, state-of-the-art research was provided, creating a comparison between our VR digital twin and the different approaches previously tested.
- To validate the claim that VR provides an edge and potential improvements over traditional web-based dashboards, a User Acceptance study has been conducted. As explored in the Analysis Section, the results imply a potential improvement in both User Experience (UX) and analysis capabilities provided by the immersive Virtual Environment.

1.4 Structure

First, Section 2 provides an introduction to the topics of virtual reality, the HLRN cluster, as well as the usage and monitoring of HPC systems. In Section 3, the related work is reviewed. After providing an overview of visualization theory, specifically of dashboards, an summary of the state-of-the-art research around VR data visualization with a focus on integrating VR with data centers is provided.

In Section 4, Methodology and Design, a taxonomy of VR headsets and game engines is first provided, comparing them against the use case of this thesis. Next, the virtual environment and its various interaction possibilities are introduced. The Unity 2D plotting library is then focused on, first evaluating existing libraries and then comparing different implementation approaches. Next, the focus is set on the live metric architecture, exploring the chosen metrics and the rationale behind each decision. Finally, the methodology behind the user study is examined, with an emphasis on the optimal study participants and the User Acceptance Test (UAT) structure, specifically on the design of all test cases.

Section 5 focuses on the technical implementation of this thesis. First, it shows the Unity project properties and configurations as well as some technical details about the virtual environment. Next, it focuses on implementing the beforementioned plotting library with a focus on the underlying math of creating the triangular faces.

After that, the implementation of the event-driven, asynchronous live metric architecture will be thoroughly explored. Starting at the query generation, a visual workflow leveraging Grafana for faster incremental progress will be presented. After that, the singleton structure powering the data fetching and processing will be shown. Following this, the data fetching, notification, and processing by the subscribers will be explored. In the end, the integration of offline data will be inspected.

Concluding the implementation chapter, the focus will be on the technical parts of the UA study. First, the procedural mock data generation will be explored in a bottom-up manner. Lastly, it will be shown how this mock data was integrated into both the Grafana dashboard as well as its VR digital twin.

Section 6, the evaluation, will focus on the results of our user study. It will begin by providing details on the participants and the study structure, which were previously discussed in the “Methodology and Design” chapter. The analysis methodology will be explained, with a focus on the statistical tests used. The study’s results will then be presented, followed by a discussion.

Lastly, in Section 7, the contributions and results will be reiterated, following by an outlook on possible future work.

2 Background

Starting with Section 2.1, the relevant background to VR is given. This includes the history of VR systems as well as the successful usage of current Head-Mounted Display (HMD) systems in various domains. Subsequently, in Section 2.1.1 a taxonomy of different VR devices is presented. Next, Section 2.1.2 defines what a game engine does and how different engines can be classified. In Section 2.1.3 it is explained how VR interactions are best designed. Here an explicit focus is put on the problem of cybersickness. Section 2.2 introduces the HLRN and the HLRN-IV HPC system, which is presented as a digital twin in the VE. Finally, Section 2.3 explains the concept of centralized monitoring systems and presents the current setup used at the GWDG.

2.1 Virtual Reality

According to [12] VR can be defined as “[...] *an interactive, immersive and realistic, three-dimensional computer simulated world.*” whereas this virtual world is referred to as the Virtual Environment.

Research into VR and its potential in many domains has been a research topic for multiple decades. Although the first HMD prototype was already developed in 1968 [13], they weren’t usable for practical applications due to the high-performance requirements and limited resources available at that time. Research first started to get traction with the so-called CAVE Automatic Virtual Environment (CAVE) setup, first developed in 1992 at the The University of Illinois at Chicago [14]. CAVE works by projecting an immersive VE onto all six sides surrounding the user.



Figure 3: The CAVE setup at the UIC [15].

Although CAVE found many applications in industry and the military, such as a flight simulator [16] [17], due to the cost as well as the spacial requirements it was not commercially viable for the retail end user. The first commercial successes were made with interactive, motion-based input devices, namely the Wii remote by Nintendo and Sony’s PlayStation Move. Microsoft’s Kinect technology, initially released in 2010 as an input device for the Xbox 360, allowed for controllerless motion sensing via depth-sensing. It also found many applications besides gaming, for example for identifying humans crossing the demilitarized zone across North Korea and South Korea [18].

The first modern, consumer-oriented, HMD and motion-based input based VR-Headset was the Oculus Rift, introduced in 2012 after crowdfunding nearly US\$2.5 million from over 9500 contributors [19]. Since then, VR found many applications in research, commercial and retail usages. In the field of medical education, a meta-analysis suggests that “[...] *students training with VR achieve better pass rates than those educated using traditional training methods*” [20]. Other applications are found in fields like construction [21], healthcare [22], and gaming [23].

2.1.1 Devices

Current HMD-based VR-Headsets can be grouped into 3 categories: *Mobile Devices*, *Standalone Devices*, and *Stationary Devices* [24].

Mobile Devices VR mobile devices are based on modern smartphones. They can be further subdivided into *passive* or *active* headsets. While passive headsets are only dumb head mounts to fixate your smartphone to your head, active headsets provide external Inertial Measurement Units (IMUs), i.e. accelerometers and gyroscopes. The most popular type of passive mobile device headset is the Google Cardboard, and the most popular active one the Samsung Gear VR.

The Google Cardboard is a smartphone-based VR platform developed by Google, and released in 2014. There are two different ways of obtaining a Google Cardboard compatible viewer; Not only can they be bought for less than 10€[25], sometimes being made out of actual cardboard, but they can also be manufactured by the end user since Google made both the schematics and assembly instructions freely available [26]. It does not contain any hardware as it only relies on the internal accelerometer and gyroscope provided by the smartphone. Despite the fact that it was developed by Google, it also supports iPhone models [27]. After selling more than 15 million units by 2019 and open sourcing the Software Development Kit (SDK), it was formally announced to be discontinued [28]. The third cardboard viewers are still available and the SDK and Unity plugins are still actively developed [29] [30].



Figure 4: An assembled Google Cardboard VR mount [31].

The Samsung Gear VR is the most popular active mobile device-based headset, developed by Samsung in cooperation with Oculus [32]. The latest version was available for a retail price of \$99 [33], although it is currently not available since it was unofficially discontinued in 2020. It has several technical advantages over cheaper passive alternatives.

Firstly, it has more precise head-tracking because it ships with its own IMU, connected via USB. Secondly, it provides a more mature user interface with the Oculus Home Environment. Lastly, VR applications have a higher scheduling priority due to the customized Android shipped with Samsung smartphones.



Figure 5: A Samsung Gear VR, smartphone mounted [34].

Mobile VR devices have few real-world applications besides some PoC visualizations, such as VR films [35] or YouTube 360-degree videos. The main purpose of mobile device VR technology is the acceleration of both user acceptance and third-party software development. Due to the near zero upfront cost, it removes any barrier of entry for anyone already owning a smartphone. This solves the VR chicken or egg problem: In order to incentivize users into buying expensive dedicated hardware, the software selection has to be comprised of many high-quality applications. But to fund those developments, the user base has to be large enough.

Unfortunately, the disadvantages of mobile VR are numerous. Firstly, even after 8 years of first-class operating system support, no good ecosystem has been developed. Secondly, all relevant hardware projects are out of production. Thirdly, while mobile device VR supports head tracking, it is not able to track positional movement, which severely reduces the theoretical space of applications. Lastly, since mobile device VR is non-specialized consumer hardware there are several hardware limitations, such as sensor accuracy, power management, or high latency [36].

Standalone Devices Standalone devices, or so-called All-In-One devices, are similar to mobile VR devices as they have the same Advanced RISC Machines (ARM) based computing architecture. In this section, the focus will be on the Meta Quest 2, since it is the current market leader with over 10 million shipped units according to Qualcomm [37].

The Meta Quest 2² was released in October 2020. It is an AIO HMD running on an Android-based OS with a fully customized UI, controlled via 2 motion-based controllers. No further tracking devices are required. The Quest 2 has the ability to run software in two ways: Either in standalone mode, where the software runs as an Android app completely independently on the device, or in passthrough mode, where it streams PC-VR software to the device. In passthrough mode, the calculations are done on the PC side; only the computed visuals are streamed to the VR device. Data Transfer can either

²Released as Oculus Quest 2

take place via Oculus Link over USB-C or via Airlink using Wi-Fi [38]. In the following, however, the focus will only be on standalone use cases since stationary setups still have better support for PC VR.



Figure 6: A Quest 2 VR [39].

AIO VR devices have found many diverse applications. In particular, the Quest 2, with its dual QLED displays with a per-eye resolution of 1832x1920 at up to 120 Hz, a Qualcomm Snapdragon XR2 as well as 6 GB LPDDR4X Random-Access Memory (RAM) [40] has more than enough resources for most VR use cases.

A major use case for AIO VR is collaborative work and VR-based communications. The biggest projects in this domain are Meta’s Horizon Workrooms [41] as part of their Metaverse developments and VRChat [42], which is popular in the gaming space. The next big application is gaming. VR gaming, even on mobile hardware, is very popular with games like Beat Saber [43] or Pistol Whip [44]. Other use cases include 3D painting [45] as well as various industrial applications [21].



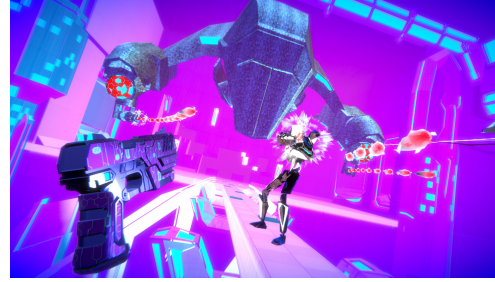
(a) Workrooms [46]



(b) VRChat [47]



(c) Beat Saber [48]



(d) Pistol Whip [49]



(e) MultiBrush [45]

Figure 7: Use cases for the Quest 2: (a) for collaborative remote work, (b) for general interaction, (c, d) for gaming, and (e) for drawing and sculpting 3D visuals.

Standalone VR devices like the Quest 2 have many advantages. First, with the comparatively low total price of 450€[40], it lowers the barrier to entry for new VR users. In particular, no high-end gaming PC is needed here, as all calculations take place on the headset itself. The compactness of the system also makes it very easy to set up, and since no external sensors are needed, locations can be changed easily. Moreover, it combines the best of both worlds: On the one hand, it can be used for PC VR applications and is cheaper than a stationary system. On the other hand, it offers a completely integrated VR-optimized experience, as all software is written for VR use only.

Systems like the Quest 2 have comparatively few disadvantages. The biggest downside is that you don't have full-body tracking since there are no sensors besides those contained in the HMD. Another issue is that PC VR via Oculus Link and Airlink are not yet matured and unstable compared to other solutions.

Stationary Devices Stationary devices are VR systems that are used only in conjunction with a PC. Currently, there are 2 leading solutions on the market: The HTC Vive Pro 2 and the Valve Index. The components of both devices are compatible with each other, and both are at least co-developed by Valve, the company behind the Steam gaming platform and the Source Engine. Here, the focus is on the Valve Index.

The Index was released by Valve in June 2019. The system, consisting of an HMD with a per-eye resolution of 1440x1600 with up to 144Hz, 2 controllers as well as 2 Base Stations, can currently be purchased for 1079€[50]. Base Stations are stationary laser-based tracking devices with a range of up to 7 meters. Both controllers are also compatible with the HTC Vive systems. Full body tracking is possible with additional Vive trackers.



Figure 8: A Valve Index [51].

The primary reasons to use PC VR are as follows:

- The application does not run on Android and cannot be ported to Unix-like systems.
- The application only runs on x86-64 processor architectures and cannot be ported.
- The application requires greater performance or tracking accuracy than is possible with an integrated processor and IMU.

Applications include graphics intensive video games such as Half Life Alyx [52], Fallout 4 VR [53] as well as high resolution modeling and CAD applications [54].

The biggest advantage of stationary devices is that most PC VR applications prioritize compatibility with them. They also have the best resolution, refresh rate, and Steam integration.

However, they also have some disadvantages. First of all, they are very expensive at a price point of over 1000€, especially since they require a high-end PC as well. For full body tracking, additional sensors are needed. Since these sensors have to be placed in the room, the space requirements are significantly higher which makes the system less portable. Finally, a wired VR system is detrimental to the immersiveness of the experience[55].

2.1.2 Game Engines

Unity defines a game engine as “Software that offers a suite of tools and features to game developers in order to build their games professionally and efficiently” [56]. More specifically, most game engines perform the following tasks, implemented in a very optimized manner, below:

- **Asset Management:** levels, models, textures and materials, engine plugins
- **Audio:** Audio file management, positional audio, audio effects

- **Build System:** Different target Operating Systems (OSs), compiling, linking, platform deployment, external assets
- **Event Handlers/Main Loop:** Core game loop that provides event hooks for developer generated content and scripts
- **Graphics:** Rendering, camera management, shader, culling, post processing, texture management, particle systems and visual effects, animations
- **Input Management:** Input events, UIs, controller support, touch support
- **Multiplayer and Networking:** Socket management, state consistency, server side validation, lag and jitter management
- **Physics:** Rigid mechanics, soft body dynamics, particle systems
- **Other:** AR and VR³, navigation and pathfinding, performance profiling

Game Engines can be grouped into different categories:

- **Low-Code Engines:** Low-code game engines enable the development of complete video games with little to no code written by the developer. Engines such as Gamemaker [57] enable the design of small 2D video games, primarily through visual programming. This is a trade-off; the more programming that is visual, the bigger assumptions about the inner working have to be made, thus reducing the customizability. The engine can also be extended by using its own scripting language, the so-called “Game Maker Language”. Other engines such as Unity or Unreal also allow the auto-generation of code through visual graphs⁴, but these are usually not classified as low-code as these are not their main paradigms.

Despite the limitations, many successful video games like Undertale [58] or Hotline Miami [59] were created with GameMaker.

- **High-Level-Language Engines:** High-level languages, defined by memory management not being manual, but instead through garbage collection, usually present a trade-off: On the one hand, memory-safe code is more secure [60] and the development cycle is faster, but on the other hand, the maximum performance cannot be achieved. The most popular engines extendible via high-level languages are Unity [9], which uses C# as a scripting language, and Godot [61], which developed its own Python-like language called GDScript.

High-Level-Languages like Unity are usually the most used engines in indie and small-scale development [62], but are also used by big game development studios such as Blizzard Entertainment⁵.

- **Low-Level-Language Engines:** Using low-level languages, i.e. languages with manual memory management and raw pointers such as C or C++, usually offer the highest performance. They are also the most customizable since game logic

³also called Extended reality (XR)

⁴Unreal allows the creation of gameplay system logic through visual scripting through their Blueprints system. Unity allows the automatic generation of shader as well as Visual Effects (VFX) logic through their Shadergraph and Visual Effects Graph systems

⁵The popular online card game Hearthstone is made in Unity

and engine core are written in the same language. C++-based game engines, such as Unreal [63] or Source [64], are mostly used by large game studios developing graphic-intensive video games.

- **Specialized Engines:** Since there is active research in the field of engines, there are also many very experimental ones, which cannot be grouped into one of the above categories. Some noteworthy projects are:
 - **OpenGOAL** [65] creates an open source version of Game Oriented Assembly Lisp (GOAL), created by Naughty Dog’s Jak and Dexter team for scripting, especially cutscenes.
 - **BEVY** [66] is a Rust based free open source engine to leverage the memory and thread safety provided by the Rust programming language.
 - **NonEuclidean** [NonEuclidean] is a game engine that allows using non-euclidean geometry.
 - **Roblox** [67] provides a platform on which developers can extend the Roblox core game through Lua scripting.

A more detailed and rigorous taxonomy can be found in [62]. Furthermore, a more detailed comparison of the engines will be given in the methodology in order to evaluate different game engines for this thesis.

2.1.3 Interaction Guidelines and Cybersickness

VR interaction is a difficult, actively researched problem. Bryson [68] describes the problem as follows: “[...] *the virtual reality interface is a completely new paradigm to which two-dimensional interface paradigms do not easily apply. This difficulty has forced virtual reality application developers to reinvent the human-computer interface all over again—a difficult task.*”. This section introduces the concepts of VR-specific interactions, VR best-practises and the specific problem of cybersickness in VR.

VR-specific interactions VR is an immersive technology. If VEs are designed properly, they can be leveraged for improved human-machine interaction. VEs promote a natural way of interacting with the computer interfaces in an instinctive manner. It has been shown that the immersiveness of VR interactions improves engagement and focus compared to traditional video communication software [69]. In order to maximize the advantages of VR one should use as many VR native paradigms as possible, such as:

- Create *digital twins* to represent the real world. VR’s greatest advantages are immersive Virtual Environments. The easiest way to allow for a more intuitive understanding is to recreate objects as interactive digital 3D models.
- Use *spatial sound* whenever possible. This is especially important for communication software since it enables multiple conversations in the same virtual room.
- Leverage the VR specific, *motion-based input methods*; do not rely on keyboards or other external input hardware. Examples of motion-based input systems are one and two-handed grab interactions, ray cast interactions, and gesture and hand recognition.

- Always *prioritize UX over realism*. Valve’s talk about VR door interaction in video games provides an example of how to design for UX [70].
- Whenever viable, *design the user interaction around objects*, not UIs.
- When possible, capitalize on all *3 dimensions for visualization*.

Cybersickness and VR best practices Davis et al. [12] define Cybersickness as a “[...] subset of motion sickness experienced by users of virtual reality where they appear to be moving in the virtual scene while actually remaining stationary. This stationary reality and the associated compelling experience of self-motion, also calledvection, is believed to underlie the condition”. Numerous symptoms can be caused by cybersickness, including eye straining, headache, disorientation, nausea, or vomiting [71].

There are three main theories about why cybersickness happens: the poison theory, the postural instability theory, and the sensory conflict theory [71].

- **Poison Theory:** The poison theory suggests that the body attempts to vomit as an evolutionary response when the user experiences hallucinations in order to remove any poison causing it.
- **Postural Instability Theory:** Postural instability theory is rooted in the idea that the human body always tries to maintain postural stability in the surrounding environment. Thus, extended postural instability leads to cybersickness.
- **Sensory Conflict Theory:** This is the most accepted theory of cybersickness [12]. This theory reasons that cybersickness is a result of conflicting information from the visual and vestibular senses, which can often occur in VEs.

Oculus [72] published an elaborative list of best practices and their rationales. The focus is mostly set on two topics: **cybersickness** and **immersion retention**.

Immersion retention can be reduced to the fact that the illusion of the Virtual Environment should be permanently preserved. All objects have to react to head movement (i.e. no static images) and the content on both eyes should be logically coherent. Everything, including UIs, should be embedded into 3D space.

Cybersickness can be avoided by allowing the user to navigate the VE as naturally as possible. According to Oculus [72], VR software should run with at least 60 FPS v-synced and a maximum motion-to-photon latency of 20ms. Avoid continuous movement if possible. For vehicles, use only forward movement. The camera should not be moved without user control. This includes cinematic shaking in collisions and so-called “head bobbing”. Furthermore, the Field of View (FOV) should never be changed, even during interactions such as aiming through iron sights.

2.2 HLRN

The Norddeutsche Verbund für Hoch- und Höchstleistungsrechnen (HLRN) [73], or the North German Supercomputing Alliance, is an association of 7 German states⁶. The main tasks of the HLRN include the operation of the HLRN-IV, the fourth-generation

⁶Berlin, Brandenburg, Bremen, Hamburg, Mecklenburg-Vorpommern, Niedersachsen and Schleswig-Holstein

distributed supercomputer system, and the provision of the associated competence network in the field of scientific computing as well as user-specific research. The HLRN-IV is the computing cluster whose metrics are used for the visualizations of this thesis. Computing time is given to universities and institutions of the federal states via tenders. The main research areas supported by the HLRN are the following [74]:

- Environmental research
- Life sciences
- Computational chemistry
- Materials science
- Fluid dynamics

HLRN-IV, the supercomputer operated by HLRN since 2018, is located at the Georg-August-University Göttingen as well as the Zuse Institute Berlin. It consists of a total of 2695 nodes, with a total main memory of over 950TB and a peak performance of around 16 PFLOPS [75].

2.3 Usage and Monitoring of HPC Systems

In this section, the usage and monitoring of HPC clusters will be focused on. Firstly, from a users perspective, it will be shown how the basic interaction with a HPC cluster works by explaining the workflow of batch job submission, which is particularly relevant for understanding the UA study. Next, the concept of monitoring systems and their advantages will be introduced. Lastly, by using the current setup of the GWDG as an example, it will explained how the different parts of a monitoring solution work together.

Job Submission To send a job on an HPC cluster, one must first connect to one of the frontend servers via SSH. These servers are not meant for computing and should not be used for this purpose, as doing so can make them inaccessible to other users and render the entire HPC system inoperative. Once connected to a frontend server, the user can load the necessary software, compile or download their own, and create a shell script with SLURM [76] commands that defines the execution order and computing resources to be used. Most HPC systems are batch systems, meaning that jobs are executed asynchronously after being placed in a specific queue according to the requested resources. After creating the shell script, it can then be enqueued as a job via the `sbatch` command, and the user will be notified via email once the job has finished executing. The email will contain the shell output of the script, and afterwards the user can pick up their processed data from the frontend server.

Monitoring In the context of data centers and HPC, monitoring refers to the practice of tracking the performance metrics of large systems in a centralized manner. This can involve tracking a variety of metrics from various data sources, such as hardware utilization metrics from the kernel, power and cooling metrics from the infrastructure sensors, data about the specific scheduling and efficiency of executed jobs, as well as specific application data of continuously running services.

Monitoring helps system administrators understand their systems by providing real-time data on the performance and health of the system. It can provide valuable insights into the system usage and user behaviour, allowing the data center to optimize the system resources and ensuring that they are meeting the needs of their users. Additionally, monitoring helps improving the availability and reliability of systems and services. For example, if a server in a data center is running low on memory, the monitoring system could alert the responsible administrators, allowing them to address the problem before it causes the server to crash, become unresponsive or otherwise reduce the User Experience.

Monitoring at the GWDG At the GWDG, the monitoring system is set up using a metrics server running Grafana [5], an open-source platform for data visualization dashboards. The Grafana dashboards get their data from so-called datasources, which are usually provided by TSDBs. A TSDB is a database specifically designed and optimized to store and manage time-series data. In the case of GWDG, the TSDB being used is InfluxDB [10].

The HPC nodes use a tool called Telegraf [77] to send the data to the InfluxDB database. Telegraf is an open-source tool that can collect and send data from a wide variety of sources, extendible via a plugin architecture. After sent to the TSDB, the data can then be analyzed visually through Grafana's Frontend.

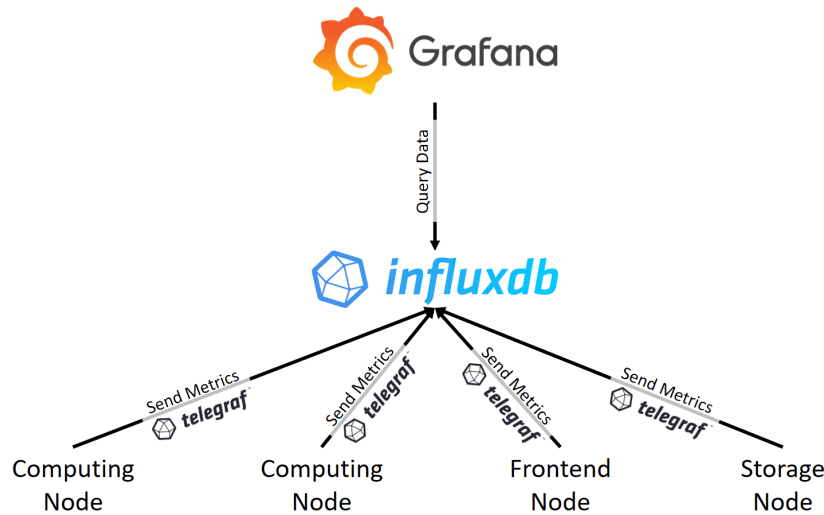


Figure 9: The metrics server structure at the GWDG

This setup is a common way that monitoring systems are set up, but there are also alternatives. For example, HLRN uses Prometheus [78] as its TSDB instead of InfluxDB. Additionally, some organizations may use cloud-based monitoring solutions like Datadog [6] instead of an on-premise solution like Grafana.

It's worth noting that the database used in a monitoring system doesn't have to be a traditional TSDB. For example, Kibana [79], which Grafana is based on, uses Elasticsearch [80] as its data store. Elasticsearch is primarily used as an open-source search engine, but it can also be used to store and index time-series data.

Next, an overview of the related work, focusing on visualization theory and the current research of VR research will be given.

3 Related Work

Section 3.1 describes aspects of visualization research and explicitly dashboard research, as these are the 2D equivalent of this thesis. Next, Section 3.2 highlights the mixed evidence on the effectiveness of VR data visualization. Finally, Section 3.3 reviews previous efforts in data center related VR projects. A classification into different subcategories is provided.

3.1 Dashboards and Visualisation

Real-time dashboards are ubiquitous nowadays, especially in the area of system monitoring and Business Intelligence (BI). Vázquez-Ingelmo et al. define dashboards as “[...] a set of (visual) resources that enable its audience to understand and/or reach insights regarding the data being displayed” [81]. Despite their ubiquity, however, little research on dashboard visualization has been done [82].

Dashboard design and dashboard evaluation based on quantifiable metrics are both difficult problems. Furthermore, it is impossible to design a dashboard that is optimal for all users; the design should depend on the use case, relevant data, and visualization skills. These are also dependent on social factors such as personal biases or beliefs [83] [84]. Thus, most literature is only providing high-level recommendations such as designing the dashboards according to the goals of your users, creating a narrative around your data, prioritizing the important information, and allowing the user to accomplish tasks to achieve their purpose [85].

In general visualization theory, most literature also consists of time-tested best practices rather than rigorous research, such as the following[86]:

- For bar charts, the y-axis must start at zero, the x-axis should have consistent intervals.

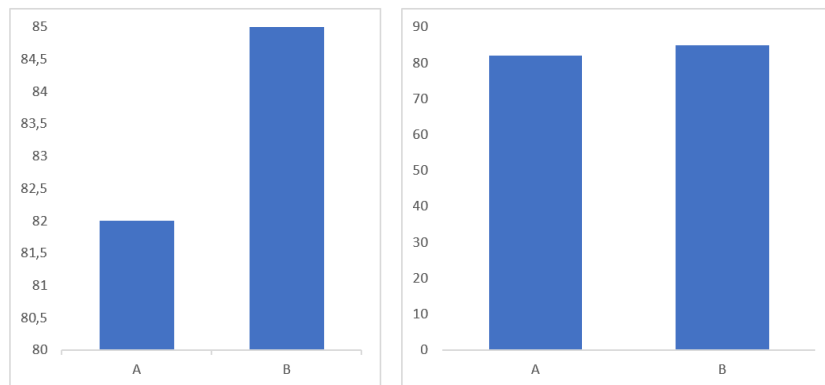


Figure 10: For bar charts, the y-axis should start at zero.

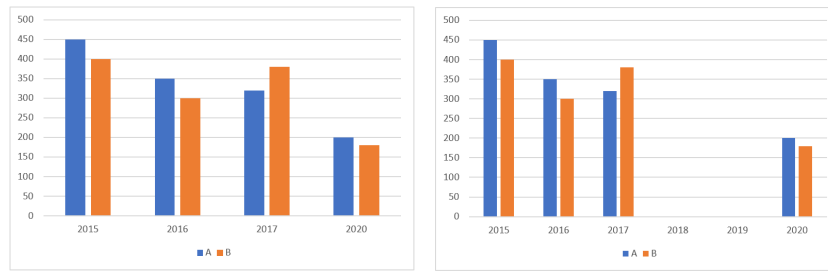


Figure 11: The x-axis should have a consistent interval size.

- When comparing data, highlight the most important information.

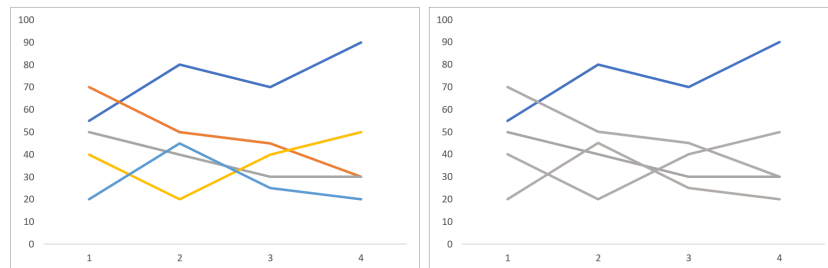


Figure 12: The most important information should be highlighted in order to most efficiently communicate the data.

- Do not use more than 6 colors. Make the colors accessible for black-white printed versions and color blindness⁷. If color maps are required, use libraries for perceptual uniformity [88].

⁷Color blindness simulators can be used in order to test the colors: [87]

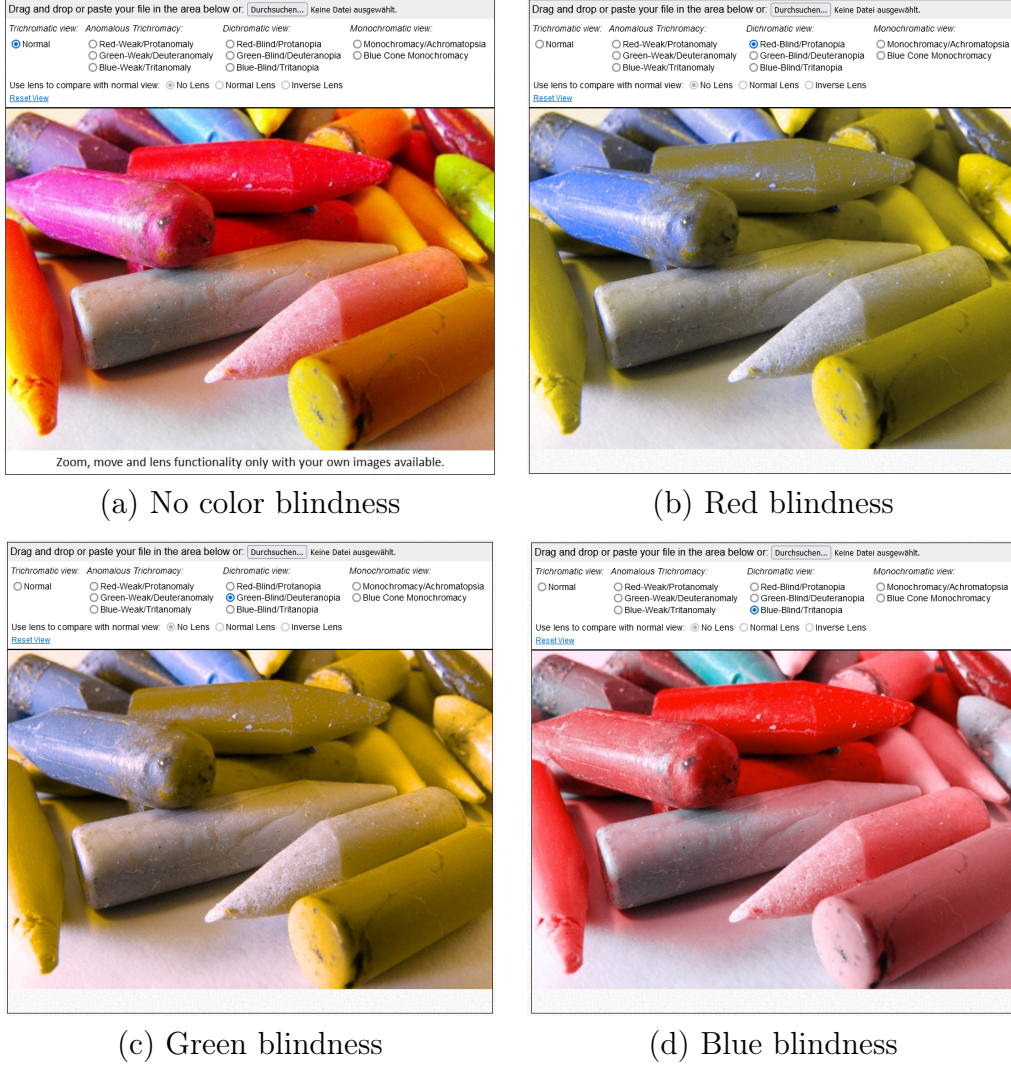


Figure 13: Example of the color blindness simulator Coblis [87].

- Use Tufte's principle of data-ink [89]. Data-ink is the ink of the graphic that represents actual information. For example, data-ink are the data points and necessary labels while non-data-ink are the frames, grids, and unnecessary documentation. One should, within reason, try to maximize the so-called data-ink ratio, which is defined as

$$\text{data-ink ratio} = \frac{\text{data-ink}}{\text{total ink}}$$

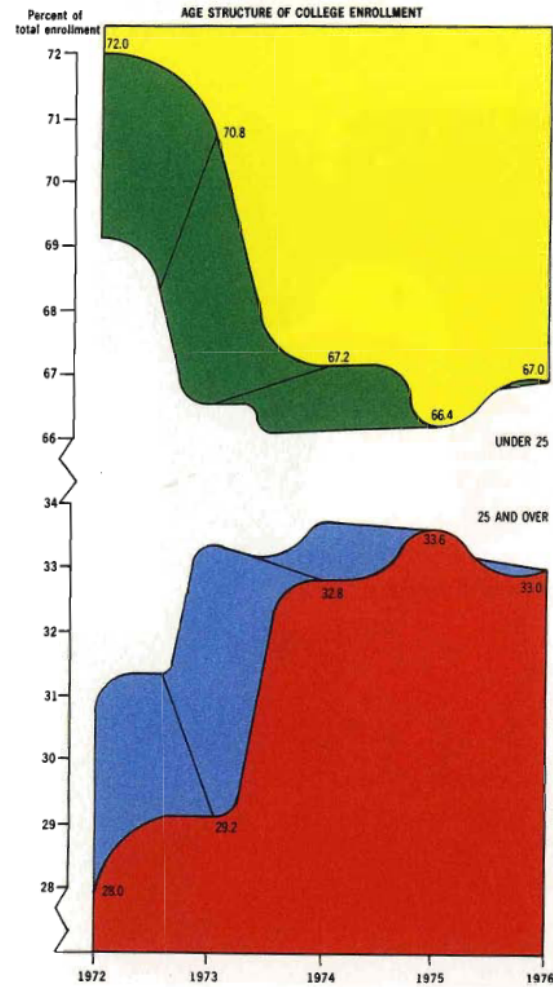


Figure 14: An example of a bad data-ink ratio. This graph contains 5 points of data [89].

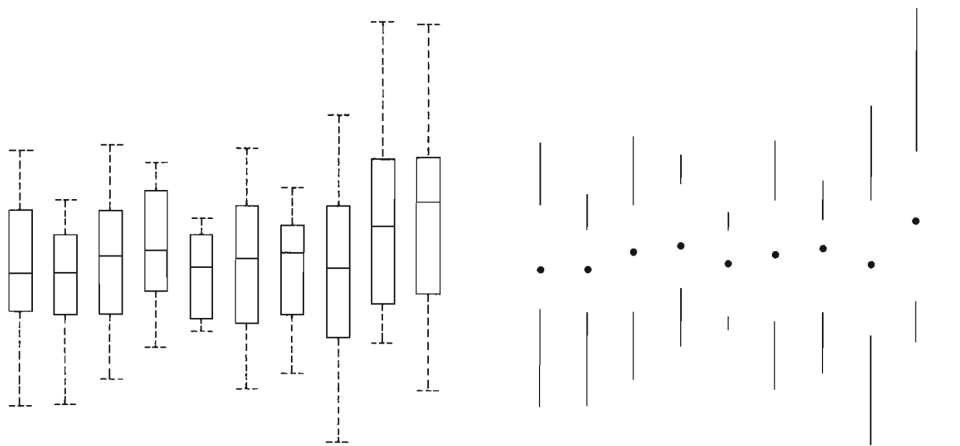


Figure 15: Tufte's redesign of the box-plot, maximizing the data-ink ratio [89].

3.2 VR Data Visualisation

Virtual Reality provides a way to enhance classic 2D visualizations through the strategic use of spatial and depth information with immersive VEs. Research on VR visualization

is rather limited, as no studies are published on most of the applications. Findings on the effectiveness of VR-based visualizations are mixed. While some papers have found better insights based on user studies [90], others have found no significant differences [91].

The biggest advantage of VR seems to be the “fun factor”. Users seem to prefer VR environments over traditional 2D non-VR ones [92]. Also, people are more satisfied overall with the experience, which makes for higher engagement with the data [90].

3.3 Data Center VR

There are a lot of Proof of Concepts (PoCs) around data center visualizations using Virtual Reality. They can be divided into two groups: Projects that use 360-degree camera footage and ones that use game engines for more interactiveness and immersiveness.

360-degree camera based setups There are many projects providing data center tours using 360-degree cameras. These are not interactive. Moreover, only the physical facilities are shown; information about the state of the server is missing. There are 2 approaches:

First, there are video-based tours. Here, a narrated 360-degree camera is used to walk through the data center while the visuals are explained along the way. Usually, these guides are uploaded to YouTube [93] [94].



Figure 16: Leaseweb 360-degree data center tour on YouTube [93].



Figure 17: IronMountain VA-1 360-degree data center tour on YouTube [94].

There are also stationary approaches that use still images with a Google Streetview-like UI for navigation [95]. Although more interactive, it is also less information dense since there is no guided narration possible.

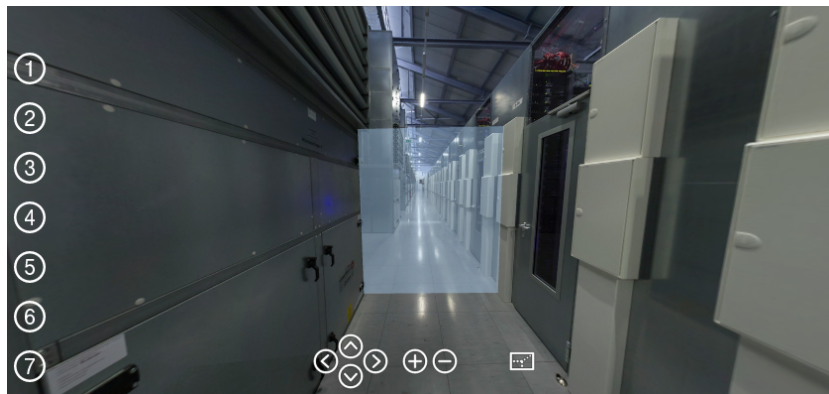


Figure 18: Manually controlled 360-degree tour through one of Hetznerns data center [95].

6 Degrees of Freedom (DoF) camera system A more immersive approach to image-based data center tours was done by Teatime research [96], a finnish based XR studio, using a 6 DoF camera setup. 6-DoF setups are sophisticated camera systems that allow estimating the image depth in order to allow motion parallax while using recorded images. Example systems are the stacked omnistereo by Thatte et al. [97], which stacked two camera rigs on top of each other and Googles “Welcome to light field” that uses a spinning camera setup for both capturing the image and approximating the depth contained in it [overbeck2017system]. Teatime research did not document the hardware used. Unfortunately, the VR software is not publicly available either. A recorded video of the software can be found on their website [98].



Figure 19: A recording of a 6-DoF-based recording of Green Mountains data center [98].

Engine-based VR systems There are many projects, ranging from commercial VE designers applied to data centers to consumer-oriented video games for building one's own PC, which enable the visualization or interaction of data centers and servers. No formal papers are written about an evaluation of their effectiveness. Most are not openly accessible. In the following paragraphs, the most important projects will be presented.

VE Configurators Axonom created the Powertrak VR Product Configurator [99], a 3D design tool built for PCVR setups such as the Valve Index or Oculus Rift. It allows for basic physical and teleportation-based movement. It is also used for the configuration of server racks, where it is possible to physically move the different servers within the rack. As of this writing, the software is not publicly available.

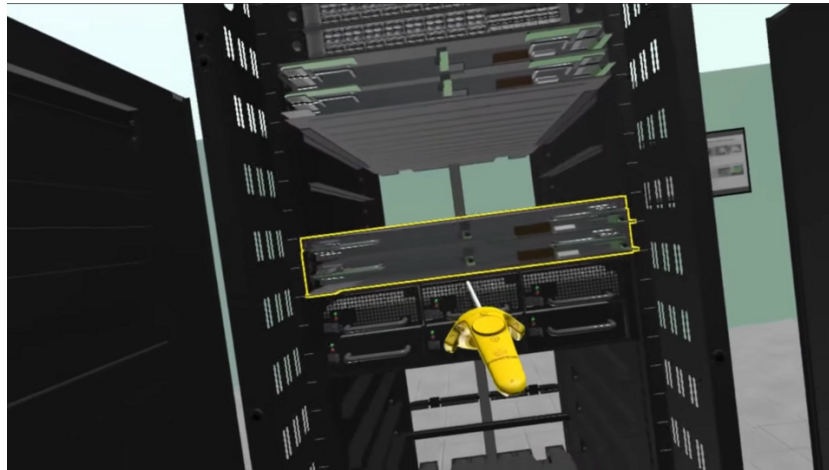


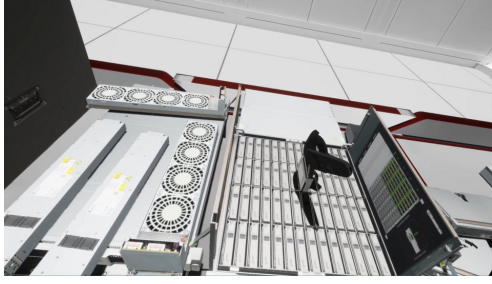
Figure 20: A VR solution for configuring a server rack [100].

Device Interaction Hypercane studios created a VR data center simulator made for the Oculus Quest AIO VR setup [101]. This PoC is mostly focused on interaction. It allows several interactions with the digital twins of realistic server hardware, such as:

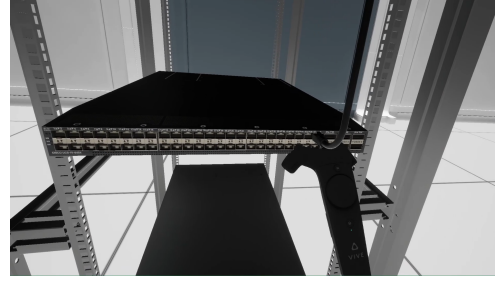
- Adding and removing server and switch components in an interactive way
- Manually connecting cables between switches

- Create guided learning environments for teaching

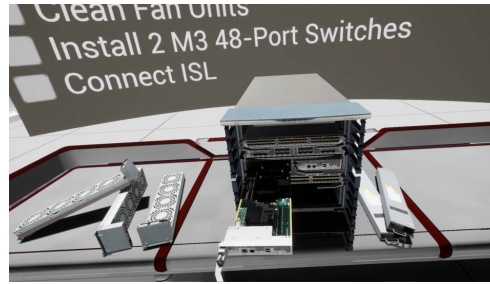
As of this writing, the software is not publicly available either.



(a) Interactively replacing storage



(b) Connecting switches with a cable system



Guided configuration of a 10 slot switch

Figure 21: The various VR native interactions provided by the simulator [101].

Data Center Planning Meta, the developers of the Oculus devices, are internally using the Oculus Quest 2 itself to design new data centers. They, together with Mortenson Construction and InsiteVR, presented the internal software at the 2022 AWE XR conference [21]. It provides a collaborative, real-time VE for all subcontractors to detect mistakes earlier. It also integrates into modern Building Information Modeling (BIM) software, hence it works with industry standard tooling for inter-company collaboration of large-scale building projects. Unfortunately, it is only for internal use and is not publicly available as well.

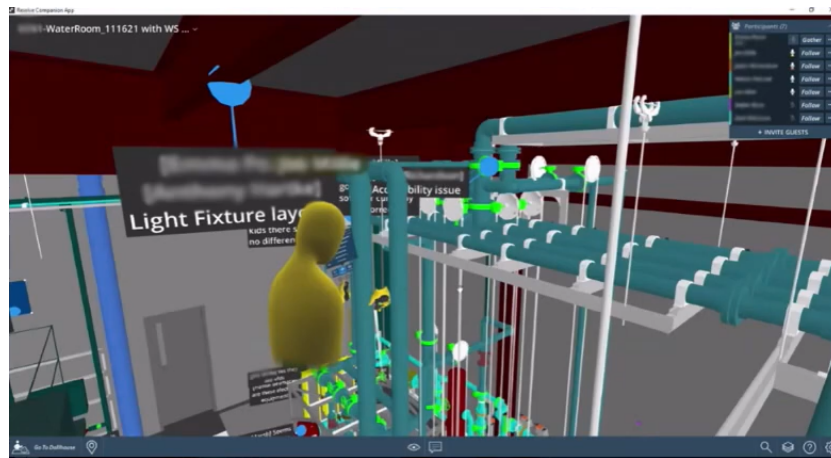


Figure 22: The interface of the VR construction software used internally by Meta [21].

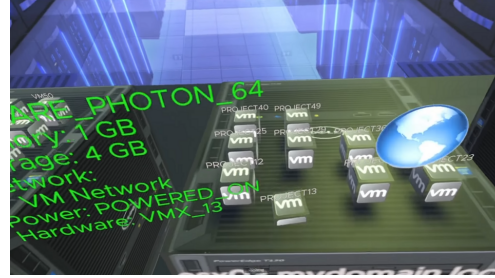
Another data center design was also developed on stationary VR-based technologies by dtm, although no public footage exists [102].

Other Lastly, there are two other projects that, although they do not fit any of the previous categories, are still noteworthy:

The **VMWare VR Datacenter Experience** is a PoC for visualizing vCenter environments. It was first presented at the VMWorld 2017 Europe [103]. It is built using Unity and targeted stationary headsets, i.e. the Windows platform. It could stream basic data from the vCenter and print it out in world space. Although discontinued, the source code is still available on Github [104].



(a) The general UI



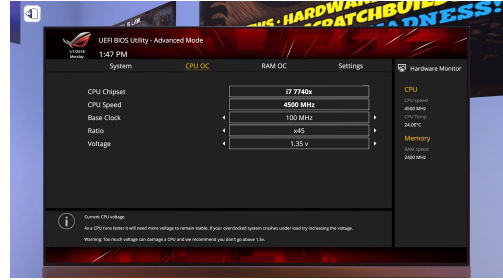
(b) Streamed data from the configured vCenter

Figure 23: The interface and streamed data supported by the VMWare VR Datacenter Experience [103].

Although not VR, the popular “PC Building Simulator” is a very detailed and elaborate simulator game in which one builds consumer PC setups [105]. It provides realistic hardware as they have many partnerships with chip manufacturers such as Intel, AMD, NVIDIA and peripheral producers such as Razer, Corsair, or SteelSeries. It covers everything from choosing a case, to connecting the components and even benchmarking with 3DMark afterward.



(a) An fully licensed example configuration



(b) Overclocking the processor via the BIOS.

Figure 24: The PC Building Simulator [105].

Although a lot of research has been done in the field of VR visualization, it was not previously tried to visualize live metrics in VR. In the next chapter, the methodology and desing behind this data visualization approach will be further looked at

4 Methodology and Design

Section 4.1 of the presents a taxonomy of various virtual reality headsets and game engines, comparing them for the use case of the thesis. In Section 4.2, the virtual environment and related interaction mechanics are introduced and the reasoning behind their design is explained. Section 4.3 discusses the design of a new 2D plotting library for Unity, beginning with a comparison of existing alternatives before considering different implementation approaches. Afterwards Section 4.4 provides an overview of the high-level ideas of the Grafana live metric architecture, followed by a detailed listing and explanation of the visualized metrics. Finally, Section 4.5 presents the methodology of the user acceptance study, starting with an description of ideal study participants and continuing with the high-level structure of the user acceptance testing, mainly focusing on test cases.

4.1 VR Technology Stack

In the background section a taxonomy of the different VR setups and game engines is provided. Building upon this, this subsection provides the rationale behind the technologies used in this thesis, particularly the VR headset and game engine chosen.

4.1.1 VR Headset

The choice of the optimal VR hardware depends on many components. The most important parts are immersiveness, the hardware specifications as well as the general UX. In addition, the three categories of the previously determined taxonomy are compared: Mobile devices, AIO devices and stationary devices.

Immersiveness: Immersiveness is the most important part of a good VE. Without immersiveness, the most essential advantages of VR are lost: focus is reduced, the virtual environment feels less realistic, and the fun factor is diminished. From an immersiveness perspective, mobile devices are fundamentally unusable. Since it only allows for head, but not positional tracking, the possibility of creating immersive VEs is acutely limited. Also, as mentioned before, the immersiveness of stationary systems are limited by the requirement of cords [55]. Immersiveness is not equal to photorealism; although stationary systems allow more graphically intensive, AAA-type environments immersiveness can also be archived with simpler art styles.

Hardware Specification: The possible quality and scope of the VR experiences are limited by the hardware specifications of the VR devices. Obviously, stationary devices have the better specifications, even if only due to the purchase price and the processing power of modern desktop devices. Mobile devices are, as already mentioned, unviable due to the lack of sensors. Thus, the primary question is how much difference the stationary hardware is compared to the AIO devices. In this case, the Quest 2 is used for those comparisons as it is currently the most sold AIO headset.

With a per-eye resolution of 1832x1920, the resolution of the Quest 2 is superior to stationary alternatives like the Valve Index with 1440x1600. While the Index's 144Hz refresh rate is superior to the Quest's 120Hz one, in reality the difference isnegligible. The result of more servere performance limitations in game development is very complicated

to quantify. Many games like Half Life Alyx [52] or Skyrim [106] are mainly thriving on their vast, sophisticated realistic environments. But for practical purposes, this question is irrelevant, as any application of this scope would immensely exceed any bachelors thesis. After all the computing power of any modern AIO device suffice for all PoCs creatable as a bachelors thesis.

End-User Experience The value of a good end-user experience should not be underestimated. Any software, no matter how immersive or otherwise beneficial, is of no value if the hardware is not accessible to end users. The higher the effort required for using the hardware, the fewer active users will exist.

Besides the aforementioned problem of positional tracking, mobile VR actually has a good UX. Both AIO and mobile devices have several advantages over stationary PC-based VR setups. The most obvious advantage is the acquisition cost. Smartphone based VR headsets, passive as well as active, range in price from \$10 to \$99 [GetCardboard] [33] and are thus very accessible to most consumer, since essentially everybody owns a smartphone nowadays. But even with a higher price of up to 450€[40], AIO devices are outstandingly more accessible then modern high-end gaming setups ranging up multiple thousands of dollar. Since all applications and libraries created for this thesis are open source, it should also be accessible to as many readers as possible.

Even more important are the spacial requirements of stational VR devices. Due to the immobile desktop hardware and the various room sensors, stationary VR environments require their dedicated room. This is especially important for this thesis. Since the software may also be shown at conferences or other events, the required hardware should be set up as easily as possible.

Conclusion: In conclusion, is was found that the AIO devices offered the best trade-off for the scope of this thesis, as shown in this comparison matrix:

	Mobile	Standalone	Stationary
Unrestricted VR movement	only head tracking	yes	cables required
Sufficient hardware	not enough sensors	yes	yes
Accessible to end-users	yes	accessible enough	high barrier of entry

Figure 25: Comparison matrix for different VR hardware

4.1.2 Game Engine

As previously mentioned in the Background Section, game engines provide out-of-the-box solutions for most of the components required for the development of VR applications. Therefore, selecting the most appropriate game engine should not be done as a mere afterthought. In this section, we will evaluate the most used publically available game engines for VR development by doing a feature-wise comparison. At first, we will look which engines are the best fit for modern VR development by looking at their tooling and VR compatibility. Next, we will compare those engines in their Developer Experience (DX), whether they have any missing features and their popularity in the development community. Lastly, we will use those insights to conclude which game engine best fits our project.

Best Engines for VR Projects: At the time of this writing, not many publically available VR engines do exist. Many companies such as Bethesda⁸ or Rockstar⁹ still use in-house engines for their flagships. Valve, while being very involved in the hardware development, has no plans of publishing a SDK for their Source 2 engine powering games such as Half-Life Alyx [109]. According to the Oculus documentation, Meta officially supports four different target platforms for their AIO devices:

- **Unity Engine:** [9] A high-level engine mainly used for small studio development.
- **Unreal Engine:** [63] A low-level game engine mostly used for graphic-intensive games.
- **Native Development:** Meta also provides the OpenXR Mobile SDK [110] for AIO devices and the Oculus PC SDK [111] for stationary devices. Those are not viable for this use case, as there is neither a need for developing a game engine nor are the resources available.
- **Web Development:** They also allow an engine-less development targeting the so-called Meta Quest Browser shipped with every Meta Quest device [112]. This is a Chromium based browser, which enables graphic intensive VR development via WebXR and WebGL, natively embedded into the webbrowser. This solution doesn't fit our use case as well for the same reasons above.

This leaves us with Unity or Unreal as our engine choices, which will now be evaluated for their DX, features and popularity among the developer community.

Developer Experience The DX of a framework can be very subjective and depend on the preference of the developer. Nevertheless, there are some parts that can be compared between Unity and Unreal. First of all, both engines enable visual programming. Unity has several visual scripting systems: The Shader graph for shaders, visual effect graph for VFX and Unity Visual Scripting (formerly Bolt) for general purpose scripting. Unreal offers the so-called Blueprints visual scripting system, which can be used to create complete games. However, visual graph-based programming can become very complex. Visual scripting is also slower than its code-based counterpart. When comparing the actual programming, the different DX can be broken down into two components: The scripting language itself and the embedding of the language into the game engine.

As mentioned in the beginning, Unity uses C# and Unreal C++ as their scripting languages. According to the 2022 Stackoverflow Developer Survey with over 70000 participants, C# is the most popular pure Object-Oriented-Programming (OOP) language with over 63% of developers "loving" the language instead of dreading it. C++, on the other hand, is more dreaded than loved [113]. Even though the survey didn't ask why C# is more popular, there are a few objective advantages of C#: By being developed by Microsoft instead of an ISO committee, language development is faster. C# has a working package management system with a remote registry¹⁰ and due to its age and

⁸Using the Creation Engine: [107]

⁹Using the Rockstar Advance Game Engine: [108]

¹⁰C++ also has many package managers such as Conan or vcpkg, but they are not part of the official ecosystem yet. What makes matters worse is that the C++ build system ecosystem is also very heterogeneous, ranging from simple Makefiles to sophisticated CMake or Bazel setups.

backwards compatibility requirements, legacy and modern C++ are basically essentially two different programming languages.

Another important difference is how the languages are integrated into the game engine. While Unity exposes the engine as an API and allows writing very idiomatic C# code, Unreal is a bit more optionated: First of all, Unreal does not support the Standard Template Library (STL)¹¹ [115]. Also, Unreal relies on a lot of macros. Both of these things increase the initial learning curve for someone who already knows C++.

Features For developing VR applications, Unity and Unreal have complete feature parity as both engines offer everything required while targeting all commonly used setups. With the Unity Asset Store and the Unreal Marketplace, both engines offer a way to buy ready-made 3D assets and other functionality. The Unity Asset Store has more content, but also exists for a longer time. Lastly, both Unity and Unreal offer their own video learning platforms.

Popularity The popularity of game engines is very relevant. The larger the community, the more learning resources and libraries exist. Thus, more applications are developed as well. This creates more jobs, which in turn results in a larger community. Here are some publicly available metrics to quantify popularity:

- **Community Size:** Nowadays, most discussions take place on Reddit [116], which is considered the modern successor of decentralized forums. Thus, the size of the subforums (so-called subreddits) can quantify the relative size of the active community. On Reddit, the Unity3D [117] subreddit has a much larger user base with over 315000 members than the unrealengine [118] subreddit with 193000 members.
- **Third Party Learning Ressources:** With over 54 million registered users and over 200 thousand courses available [119], Udemy is the most popular Massive Open Online Course (MOOC) platform. On Udemy, more MOOCs about Unity exist, with around 6500 results for “Unity” and 3300 for “Unreal Engine”.
- **Commercial Game Publishing:** According to the research done by user /u/justkevin on the /r/gamedev subreddit, Unity was the most used engine of the 50 most popular Steam games of 2020 [120].
- **Indie Development:** Due to the smaller barrier of entry and their many game jams, many indie developers publish their Games on itch.io. According to their statistics, Unity is the most used engine throughout all projects [121].

Conclusion: Concluding, Unity was chosen for the game engine of this thesis. Although the popularity and consequent variety of tutorials is very convenient, the primary focus was on the DX and development speed: since this work is only a PoC, the asset store as well as faster development time are more important than the ability to create graphically elaborate experiences for which the time frame would not suffice nonetheless.

¹¹Although, as pointed out by the Author, STL is not a formal definition provided by the C++ committee. For further reading see [114].

4.2 VR Environment and Interaction

In this section, the VE will be introduced. From a top view, the layout is structured as follows:

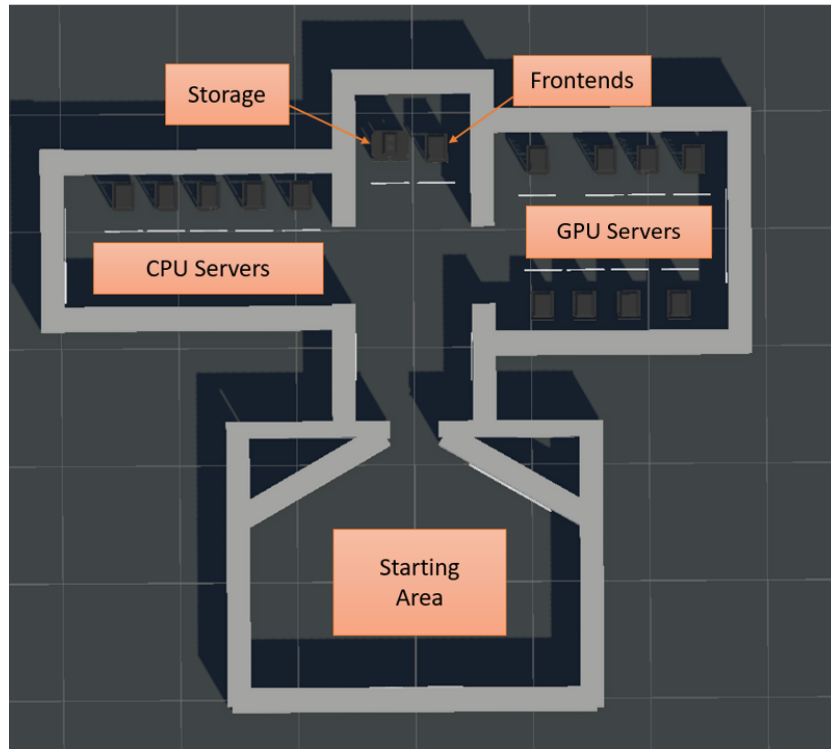


Figure 26: A top view of the VE layout

As seen in the figure, the VE was split into multiple rooms. This has multiple advantages. Firstly, it is less overwhelming for the user foreign to both analyzing metrics and VR in general. Secondly, it intuitively highlights the difference between the node types using the spatial features of VR. Lastly, it insreases the amount of exploration available. The rooms are split into the starting area containing the overview, the CPU server room, the GPU Server room and the floor containing the storage and frontend servers

In the starting area, the overview can be directly viewed when spawning. Here, a gauge chart is used in order to not overwhelm the user with a more complicated time series data visualization.

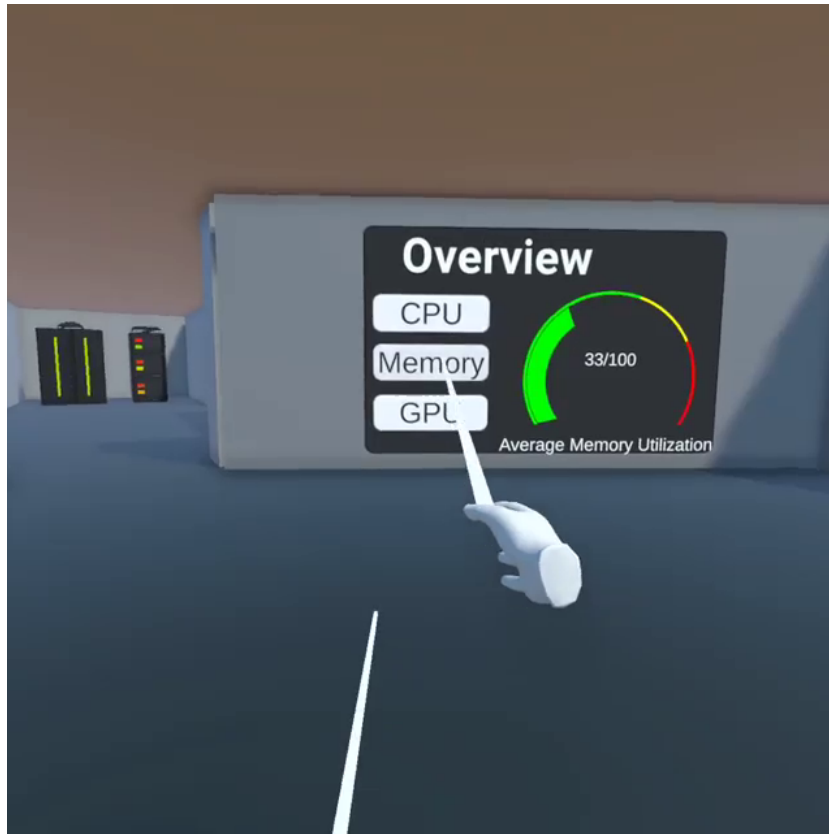


Figure 27: The overview in the starting view and the possible interaction provided.

As seen in the picture, the controller movement is visualized using animated hands, also known as “hand presence”. The animation of the hands is based on the buttons pressed. Each ray has a ray extending from it outwards, which is used for teleportation and other interactions with the VE. The general button bindings, which are the same for both controllers, can be seen here:

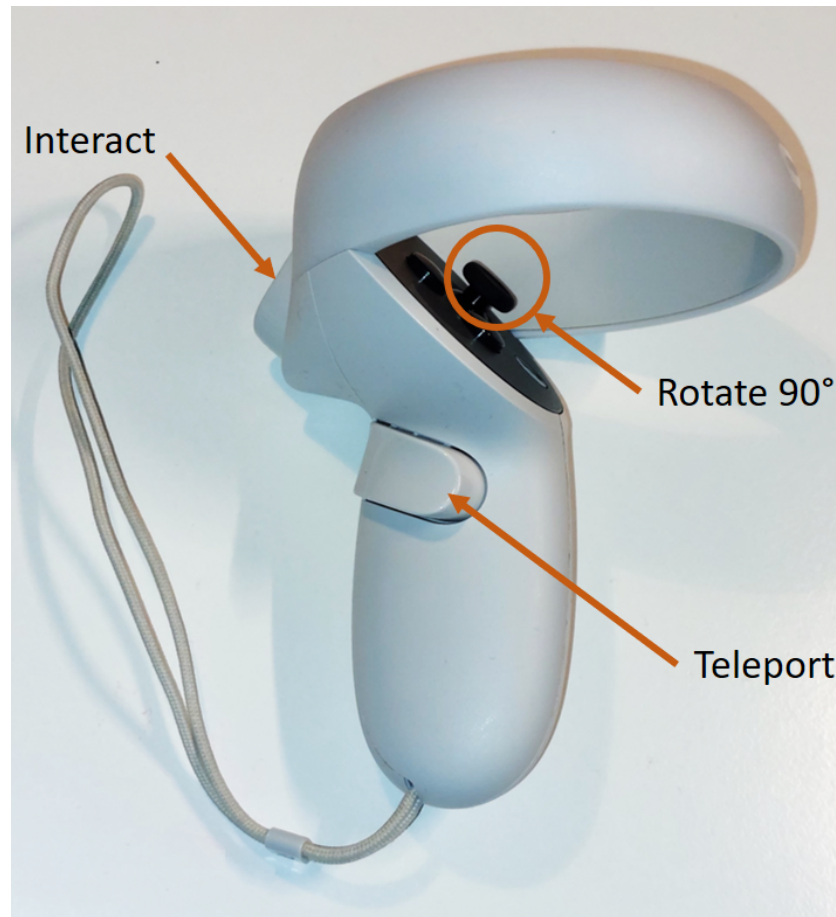


Figure 28: The button mapping of the Oculus Quest 2 controller used.

The turn stick works in a discrete manner, turning 90 degrees at a time. This is important; continuous turn movement, as explained in the background, causes cybersickness. The button based turning can also be used to maximize the usable space in the VE as it enables rotating the virtual space in order to move in the direction correlating to the most available physical space.

The rays can be used for all dynamic interactions, as well as teleporting by pointing it towards the floor. The concept of teleportation is used to fully utilize the possibility of large VEs and enable the structure of multiple rooms. It also, even more important, allows for completely stationary setups, making the application as portable as possible.

The CPU and GPU rooms contain multiple servers, as well as an tabular overview, listing one metric at a time, showing all hosts, containing a line plot of the recent utilization¹².

¹²The plot shows the last 7 data points, representing the last 210 seconds



Figure 29: The CPU and GPU server rooms

Lastly there are, besides the previously already shown overview, multiple ways the data is visualized and can be interacted with:

The server lamps indicate the current load. Servers without a GPU have two lamps, indicating the CPU and memory utilization of that node. The color is based on a green-yellow-red color gradient. This is meant as an quick overview as one can just glance in the room, skimming over each server.

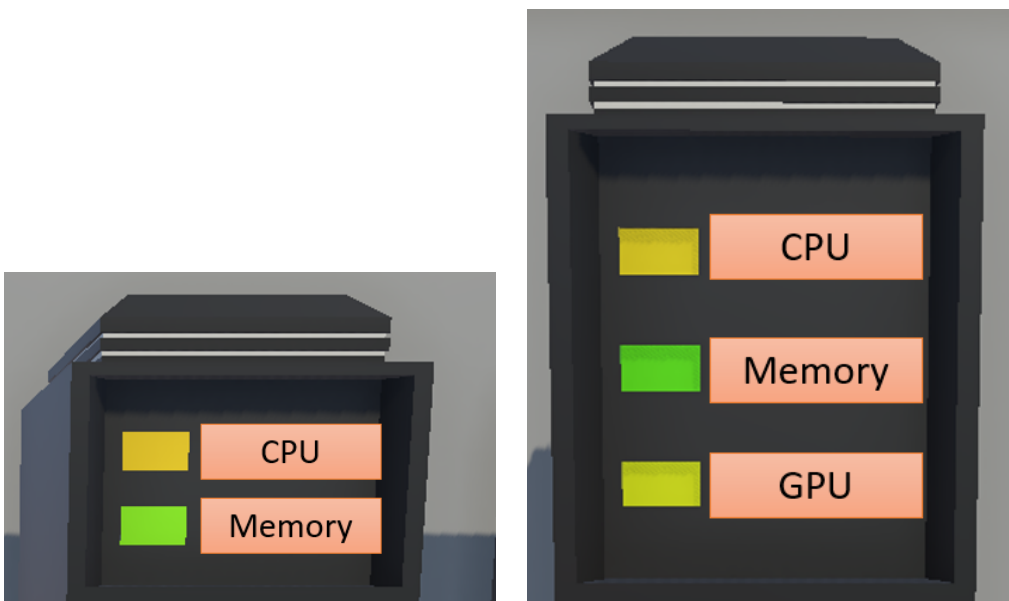


Figure 30: The lamps of a CPU and GPU node and their related metrics

The beforementioned tabular overviews can also be interacted with. By using the “Switch” button, it is possible to change the displayed metric. Furthermore, the ray can be used to scroll through the entries with a smartphone-like gesture.

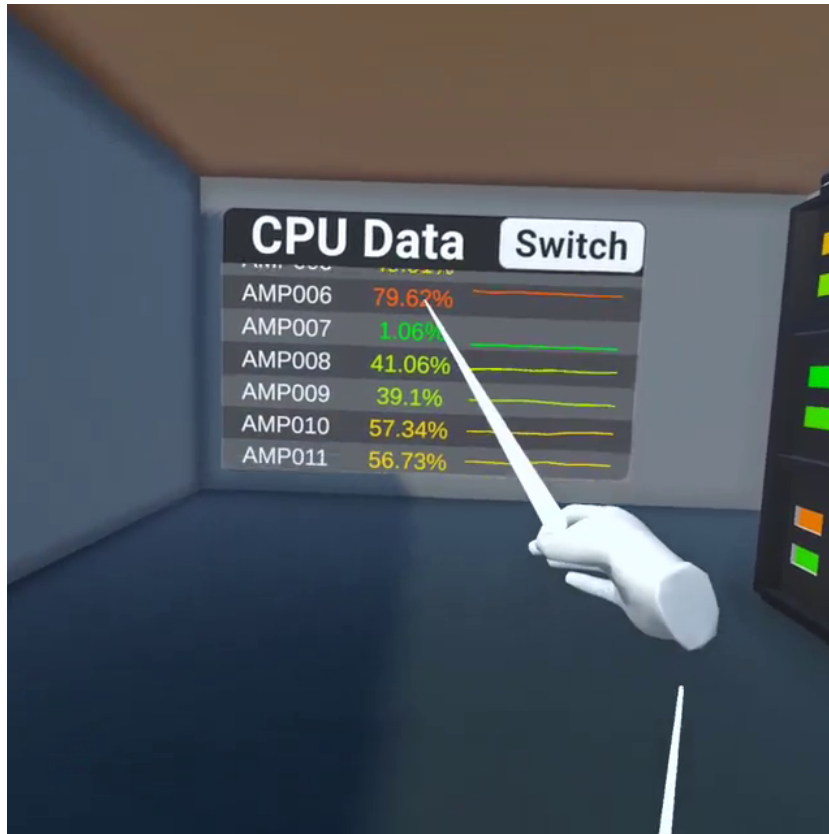


Figure 31: The tabular data overview, interacted with by scrolling using the right ray.

Lastly, each server has its own UI, which can be toggled via ray interaction on the server rack. In this menu, for each node and each available metric, the last 30 minutes of utilization are plotted. Note that the UI is always rotated towards the user, enabling the use from every angle.

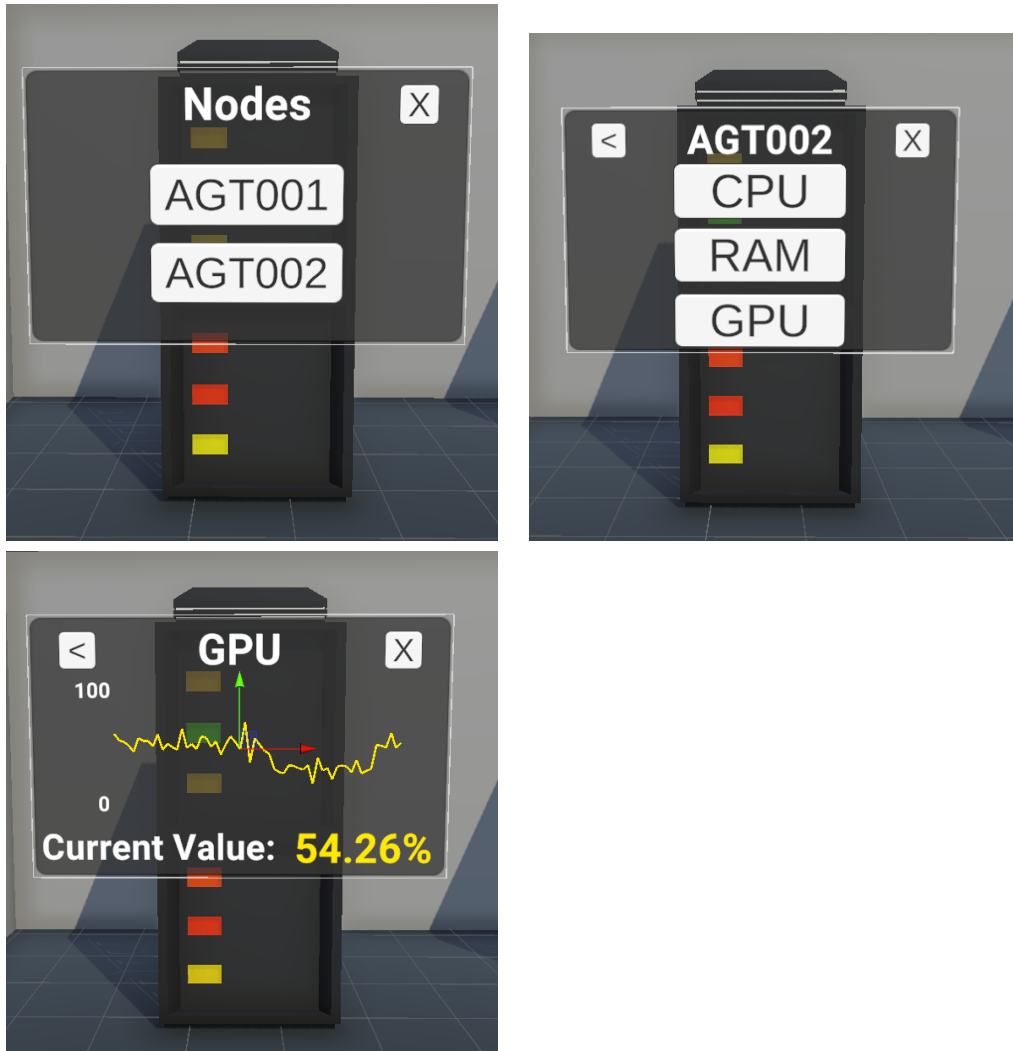


Figure 32: The server UI of a GPU rack.

4.3 2D Plotting in VR

Although this thesis uses Virtual Environments for their immersiveness, 2D graphs still have their *raison d'être* for two important reasons: First and foremost, users are most familiar with traditional 2D graphs, which makes them easier to read. What is more, some visualizations do not benefit from higher dimensionality. For example, it is not clear that more insight is gained when correlating both CPU and RAM usage together in relation to time; in this case, the higher dimensionality is not worth the additional visual complexity.

This section starts with an overview and evaluation of already existing 2D plotting libraries compatible with 3D Unity environments and why it was chosen to write a new plotting library. After that, all possible approaches of implementing a UI library will be compared.

4.3.1 Evaluation of Existing Libraries

There are many different plotting and graph implementations available on the Unity store, such as “Graph and Chart” [122] or “Awesome Charts and Graphs” [123]. From a functionality perspective, both of those libraries would suffice for any visualization used.

Unfortunately, both are non-free software licensed by the Standard Unity Asset Store End User License Agreement (EULA) [124]. To cite Section 3.5 of the EULA:

*“Unless you have been specifically permitted to do so in a separate agreement with Unity and except as permitted under the Unity-EULA, you agree that you will not use, **reproduce**, **duplicate**, publicly display, publicly perform, copy, modify, adapt, translate, prepare derivative works of, **distribute**, transfer, license, sublicense, rent, lease, lend, sell, trade, resell, or otherwise commercialize or monetize any Asset that you have licensed from the Unity Asset Store for any purpose.”*

This means that once one uses a library from the Unity Asset Store that is not publicly available, it is not possible to open source any work depending on that library. This is against the spirit of reproducible research; hence commercial plotting solutions are not viable in the context of this thesis.

Unfortunately, no open source 2D plotting libraries exist for Unity yet. Thus, a new plotting library called Plotty was created, which is publically available on Github, freely licensed with the MIT license [125]. It currently supports bar, bar and gauge plots.

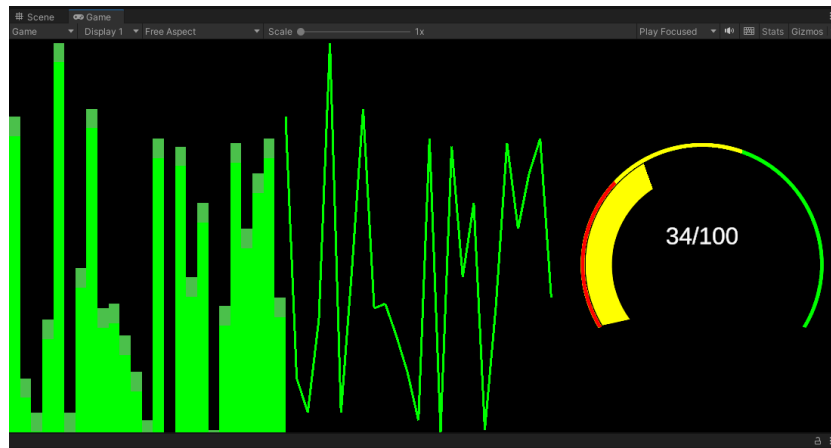


Figure 33: All plot types supported by Plotty

4.3.2 Implementation Approaches

There are several approaches to rendering line plots and bar charts on two dimensional canvas in 3D space. During the libraries creation three different approaches were compared:

1. **Image-based:** The bars and lines are each instantiated as a transformed combination of simple images. For example, a white rectangle would be a white texture with transformed x and y scales. A circle would be a square image with transparency around the circle texture. This was already done as a YouTube video tutorial series [126], although the underlying licensing is unclear.

This is clearly very inperformant, especially since it solely runs on the CPU. It is also very unmaintainable and inflexible since the graph color is based on the texture used.

2. **UIVertex-based:** The `UIVertex` class is used internally by Unity to render various components such as fonts on a canvas. It is possible to use the same APIs to manually generate triangle meshes on a canvas as documented here [127]. This is way more performant, although still computed on the CPU.
3. **Shader-based:** The last solution would be to render the data on the graphics card as a shader. Although not used for general plotting, Graphy [128] is as an open source library leveraging GPU shader to plot stats such as the FPS or memory usage.

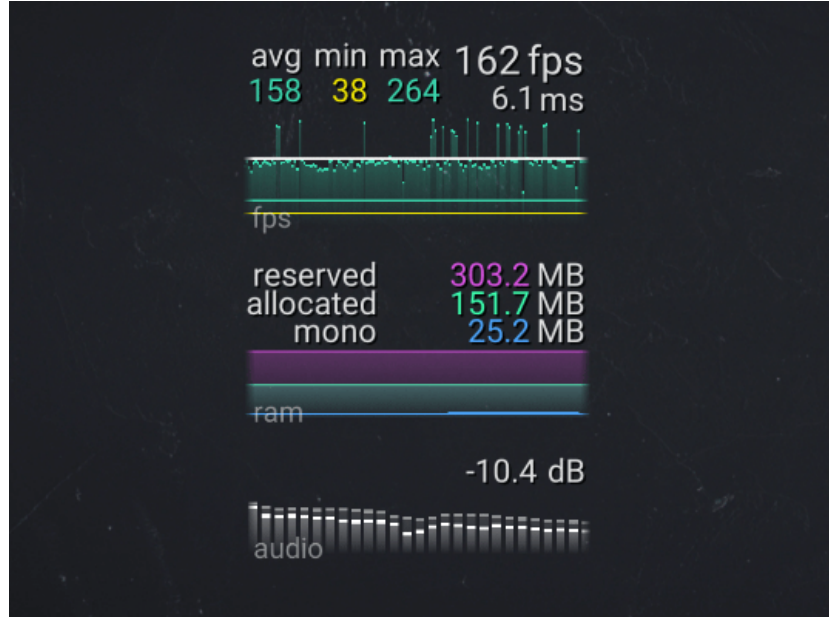


Figure 34: GPU based 2D rendering done by the Open Source FPS counter Graphy [128]

Although the most performant approach, it is also the most complicated one as C# can not be used to write shaders. Unity supports their own shader language called Shaderlab [129] as well as Microsofts High Level Shader Language [130]. Unity also supports raw GLSL [131], although this is not recommended as both other languages get transpiled into optimized, target-dependent GLSL as well.

In this thesis, we chose the `UIVertex`-based approach, as it provided enough performance and maintainability while not requiring another programming language. More on the technical details can be found in the corresponding Section of the Implementation chapter.

4.4 Live Metrics

As previously mentioned, metrics can be used to gain insights and develop an understanding of a given HPC system. For example, one can understand the availability, reliability and popularity of the single components and even infer basic user behaviour. Additionally, instead of using sophisticated dashboards that require in-depth knowledge of the cluster structure and topology, users can get an better intuition connecting the metrics to the physical cluster nodes by leveraging the spacial features of virtual environments.

The canonical way of accessing the live metrics would be to query the data directly from the TSDB. However, due to administrative restrictions, access to the data source itself was not possible. Therefore, the decision was made to query the data from Grafana.

We used Grafana’s data source proxy API [132]. When making an API call to Grafana, the selected database name and the query are passed as GET parameters. Once these parameters are passed to Grafana, the query is executed by the TSDB using the credentials that Grafana has been configured to use for authentication with the InfluxDB. This allows the client to authenticate with their Grafana API credentials, rather than having to provide credentials for the underlying database.

There are two main advantages to this approach. Firstly, it makes the API endpoint more data source agnostic, since as mentioned before, Grafana can be configured to use a variety of databases. Secondly, it simplifies the process of designing and executing queries as one can visualize them beforehand using Grafana’s UI. This will be further explored later. It should be noted that, even though we are using Grafana as the API endpoint, we still need to write InfluxQL calls in order to query the data from InfluxDB. InfluxQL is one of the query languages for InfluxDB. This means that, although the API endpoint itself is data source agnostic, the specific query only works with InfluxDB 1.7 or lower.

In order to properly be usable for presentations in a portable setting, the two following problems have to be solved:

- One potential problem that could arise when trying to access live metrics via the Grafana API is the possibility of the client not having internet access or the metrics server being otherwise inaccessible. To address this issue, different application builds were made that are able to use mocked data instead of the current metrics, creating the illusion of live data metrics.
- Another issue, especially around studies and presentations, is that an insightful visualization requires good live metrics. In some cases, the data could be such that it is either too hard to analyze for the average HPC user or one that it does not provide any valuable insights. To solve this problem, the offline builds use artificially generated mocked data instead of old replays of actual data recorded from the HPC cluster. This solution was also used for the user acceptance study, and will be discussed in further detail in the “Implementation” Section.

4.4.1 Metrics Used

Unfortunately, at the time of this writing, the HLRN-IV cluster could not be used as a data source for this server room digital twin. This is due to two reasons:

- The HLRN has its own Grafana and Prometheus [78] based monitoring solution. The accessibility of this server is, from a networking perspective, more restricted than the GWDG’s central monitoring setup. Especially, the only way to connect to this server outside of the GWDG’s internal network is to use a Virtual Private Network (VPN). The two available VPN solutions are either Cisco Anyconnect [133] or Wireguard [134], both of which are impossible to use on an Oculus Quest without rooting the device.

- Additionally, the HLRN monitoring setup is still in-progress. While the infrastructure-related metrics such as rack temperature, air cooling or power usage are already properly integrated, the computing nodes do not have any clients such as telegraf installed yet. Thus, no utilization metrics are currently accessible.

Instead, we use the SCC. The SCC [135] is a local compute cluster used by the University of Göttingen as well as several Max-Planck-Institutes. It offers a shared storage and several node types. It is currently distributed across the old headquarter as well as the Modular Data Center (MDC).

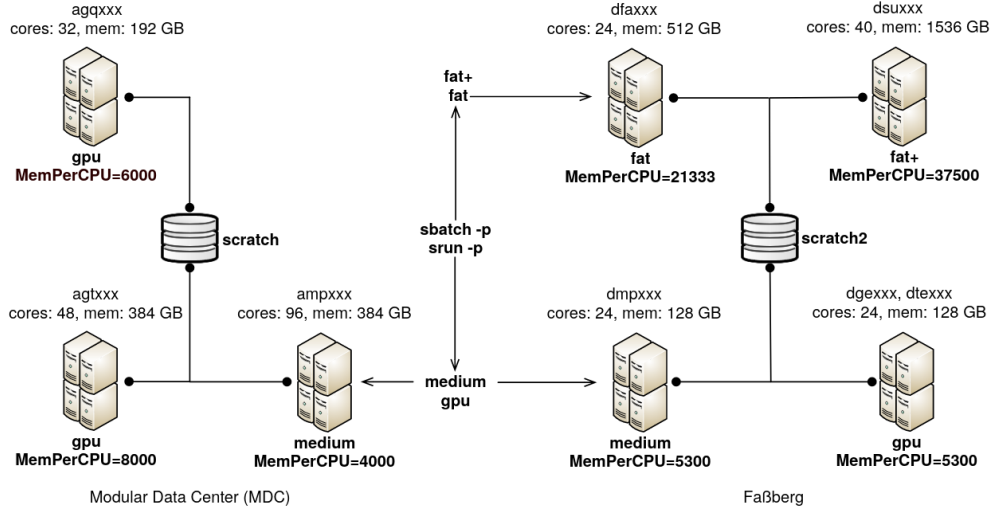


Figure 35: The SCC cluster [135]

Too keep the digital twin more accessible and have a reasonable amount of data, it was chosen to only use a subset of all available SCC nodes. To keep the physical topology connected, it was chosen to only use the nodes set in the MDC. The following nodes were chosen:

- To begin with, we chose to use include all frontend nodes. Those were the nodes the HPC user is directly interacting with, doing things such as loading modules, compiling software and configuring jobs. Those servers include the **gwdu101**, **gwdu102**, and **gwdu103** nodes. For those nodes, we chose the current CPU and Memory utilization.
- Also, we chose to include all nodes from the MDC that contain GPUs. This allows for a better variety of metrics visualized. Those nodes are the **agt001-agt002** as well as the **agq001-016**. Aside from the CPU and Memory utilization, we also visualize the GPU usage.
- Moreover, we included a subset of computing nodes that do not contain a GPU. Those are the so-called **amp** nodes at the MDC, of which there are 96. Since that many nodes are hard to comprehend in a visual setting, especially compared to the way fewer GPU nodes, it was chosen to only include the **amp001-amp015**. We also use the current CPU and Memory utilization.
- Lastly, we decided to include the storage utilization of the **SCRATCH** share.

In order to make the data more accessible and easier to understand, it was normalized to percentages. This is especially useful for users who may not have extensive knowledge or experience with HPC systems and were only previously exposed to consumer scale hardware. By presenting the data as percentages, the data is simplified from multiple CPUs with many cores to a single-dimensional value representing the utilization of the current node.

4.5 UA Study

The UA Study consists of a UAT and the corresponding analysis. The User Acceptance Test verifies whether a given application satisfies all use cases and can be used by the expected end user. In this section the focus will be on the UA study design. First, it will be discussed how the study participants were selected. Next, the general structure of the UAT will be examined. Finally, the test case design will be further explored.

Study Participants The UAT should be done by the future users. On the one hand, if a study participant is completely unfamiliar with the concept of computing clusters, they will not be able to understand the visualized data. They should be at least knowledgeable enough to understand the value of the virtual environment and the insight it provides. It should be possible for them to grasp and have a rudimentary understanding of the visualization, being able to do fundamental analysis about the current system state. Lastly, they should also be able, after a short introduction, to navigate and use a traditional dashboard software such as Grafana. Otherwise, the comparison between the VR approach and traditional, web-based dashboard technologies is not possible.

On the other hand, the participant should not be a seasoned system administrator as they are not the expected target audience. The main advantage of this digital twin is the visual intuition it provides over traditional dashboards. That it can provide insights over how the HPC system works and how it is structured by mapping the live metrics to the physical server topology. Furthermore, moving around in a VE would be too much friction for everyday usage. System administrators are used to read graphs and do not need the spatial information provided by the virtual environment because they have already understood the cluster structure.

Thus, it was chosen to base the study on computer science students which are at least in the third semester of their undergraduate degree. This way, they were already exposed to the concepts of computing, programming, servers, and have the mathematical background to rudimentary analyze time series graphs. Lastly, they do not have a regular experience with performance monitoring or the GWDG HPC cluster system.

UAT Structure The structure of the UAT is as follows:

- To begin, a simplified version of the HPC cluster and batch job system was introduced, along with an explanation of the VR navigation and embedded metrics in the virtual environment.
- The participant was then asked to complete a Pre-Questionnaire, which gathered information on their age, study major, and exposure to topics such as server main-

tenance, VR, and remote computing infrastructure as both familiarity and novelty may influence the results.

- The participant was then asked to participate in four different test cases, two of which were conducted in VR and two of which used Grafana dashboards.
- Lastly, the participant was asked to self-evaluate both technologies in order to provide feedback on their experiences.

Note that the entirety of the UAT can be found in the Appendix.

Test Cases As mentioned before, the four tasks were split into two tasks using web dashboards and two tasks using VR visualization. While the tests were slightly different, they all centered around rudimentary HPC cluster analysis based on the utilization metrics provided. To mitigate the potential of study participants becoming more accustomed to the user interface or task at hand, the software was switched after each test so that neither Grafana nor the VR digital twin was used twice in a row. The order in which the VR and web versions were introduced was also anticipated to potentially affect the results. As such, the starting order was alternated after each user. In the test cases, live data was not used. This approach allowed the data to be deterministic and the results to be independent of the current cluster load. Additionally, in order to design better test cases and facilitate easier data analysis, the data displayed was generated artificially. The appendix contains the data used in the various test cases. The participant was instructed to think out aloud while analyzing the data.

The the first two test cases of the User Acceptance Study were designed to introduce the participant to the VR and dashboard user interfaces. The first test case, “Are the servers used optimally?”, was designed to familiarize the participant with the first user interface. Mocked data was generated in such a way that the servers were not used in the most efficient manner. The frontend servers had high load, implying that they were used for computation instead of just cluster interaction and job configuration. Also the GPU clusters were had a lot of jobs only utilizing the CPU, wasting valuable computing resources by allocating those nodes.

The second test case, “How does the scheduler work?”, focused only on the nodes without a graphics card to simplify the task. The participant was explained that when a job is removed from the queue, it must be assigned one of the nodes in the pool. The task was then to analyze the metrics and infer how the scheduler works. The data was generated in such a way that the scheduler works like the following pseudocode:

```

1 for (int i=0; i<N; ++i) {
2   if (server_free(server[i])) {
3     return server[i];
4   }
5 }
```

This test case was designed to introduce the participant to the other user interface.

The third and fourth test cases were the same task, called “Long Term Decision Making”. The formal task description was:

Your department got some funds and decides on whether to upgrade the current computing hardware. The hardware should just get upgraded if it is mostly fully utilized.

Find out if there is a need to upgrade the current hardware. If there is, find out which type of servers are most needed.

After being introduced to both user interfaces in the previous test cases, this task was meant to create a direct comparison between the dashboard and VR versions. The first data set was designed in such a way that the GPU nodes were under high load, with a high GPU utilization. The second data set was designed in such a way that all resource types were used in a balanced way, with no need for hardware upgrades. All mock data can be found in the appendix. This allowed the participant to observe the differences between the two data sets and draw conclusions about the need for hardware upgrades.

In the next chapter, the focus will be on how the digital twin and UA study have been implemented.

5 Implementation

Starting with Section 5.1 the implementation details of the unity project and the virtual environment are being presented. Section 5.2 focuses on the implementation and underlying math of writing a plotting library. The implementation of the Grafana-InfluxDB-based live metrics system is discussed in Section 5.3, including the database query design, asynchronous event streaming architecture and the offline mode. Finally, Section 5.4 delves into the technical details of the user acceptance study, including the mock data generation, C# code generation and Python-based metric streaming client.

5.1 Unity and Virtual Environment

In this section, we will first describe the unity setup, focusing on the packages used and the project preferences set. After that, we will focus on a few implementation specifics related to the VE.

For this setup, the Universal Render Pipeline (URP) was used, which offers improved performance and more configuration options compared to the old Unity rendering pipeline. Since we are on an embedded device with computational constraints and do not have the resources for highly detailed models and textures, we did not use the High Definition Render Pipeline (HDRP). In order to get the URP working with the Android target, we explicitly used OpenGL ES3 instead of OpenGL ES2. For the build, we chose to use Unity's IL2CPP instead of Mono. This means that our C# scripts were transpiled to highly optimized C++ and then compiled to native code using the Android Native Development Kit (NDK), rather than shipping the Mono C# runtime. Furthermore, we chose high optimization levels, which means that we get faster code and smaller builds while increasing the build time.

The app has not been deployed into the Unity store and must be sideloaded by 3rd party software like Sidequest. For the VR components, the newer manufacturer agnostic OpenXR framework and Unity's XR Interaction Toolkit are being used. The use of OpenXR also requires the use of the new action based input system instead of the old controller based one. This means that the controls are mapped to actions such as "Activate" and "Select," relying on the VR manufacturer to create the action to controller mapping.

For the hand presence, we used the hand models created by Meta [136], converted and animated by Valem [137]. Furthermore, we built the room geometry using Unity's ProBuilder package. While the CPU and GPU servers were modeled for this thesis, the storage server is from the ipoly3d server package [138]. Here, the faces representing the lamps are different objects in order to simpler change their material.

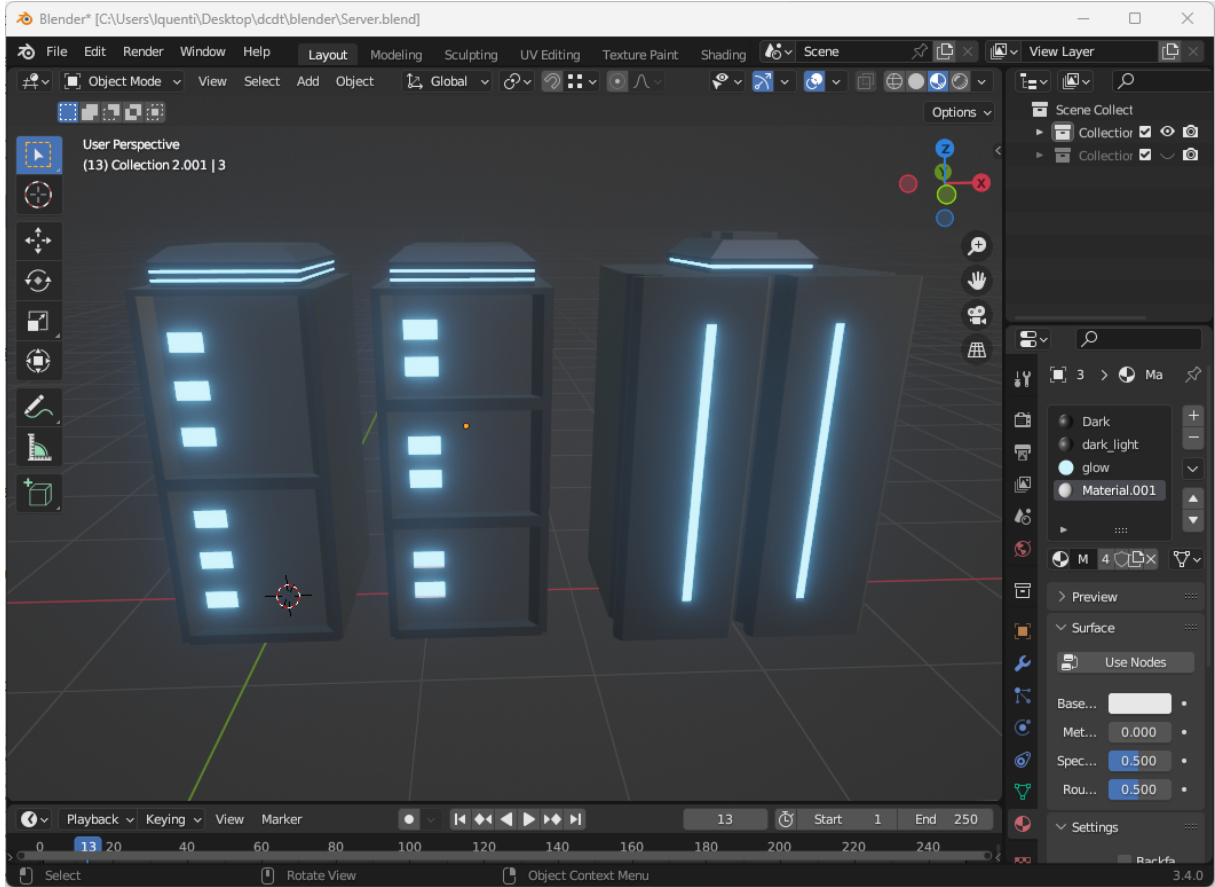


Figure 36: All server models in Blender.

5.2 Plotting Library

As previously stated in the methodology, the 2D plotting library Plotty for Unity was designed and implemented as part of this bachelor thesis [125]. Plotty supports three different plot types: Line charts, bar charts and gauge plots. For convenience, prefabs are provided. In this section we will look at the implementation details in more depth.

Line and bar chart: The basic inheritance structure can be seen in Figure 37. `Awake` and `OnPoulateMesh` are overwritten methods provided by Unity’s `MonoBehaviour` [139] and `Graphic` [140] respectively. This way, the Unity interaction is provided by the abstract base class so that the subclasses only have to implement the drawing logic through `DrawAll`. In order to provide a “Sliding Window”-look a FIFO queue of fixed length is used to store the most recent n measurements.

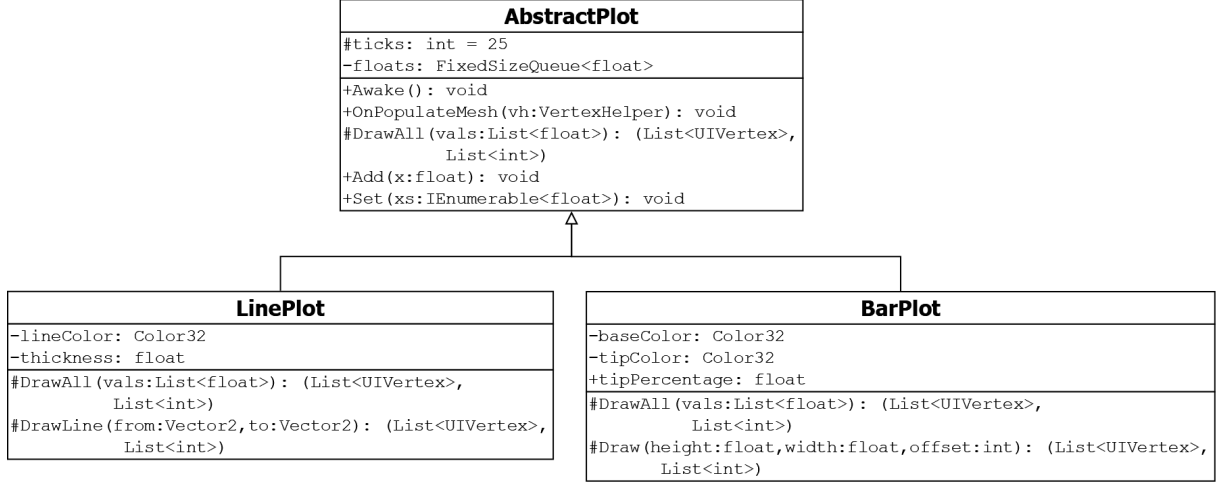
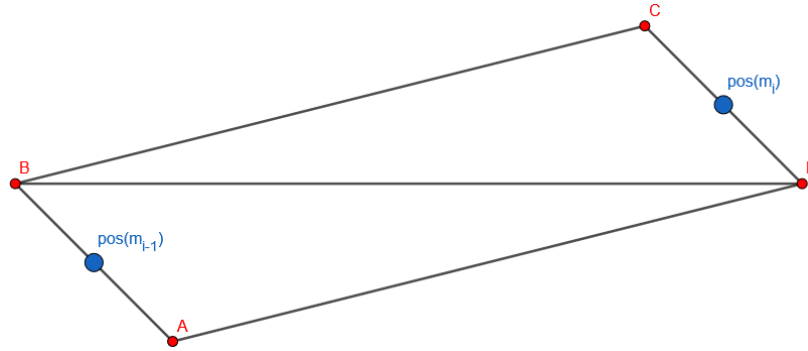


Figure 37: The UML diagram of Plotty's line and bar chart classes

Let c_x, c_y be the width and height of the canvas. The position of the i -th measurement m_i in the queue of length n is calculated by

$$pos(m_i) := \begin{bmatrix} i \cdot (c_x/n) \\ (\max\{m_1, \dots, m_n\} \cdot c_y) \cdot m_i \end{bmatrix}$$

Plotting a line works as follows: Let m_{i-1} and m_i be two measurements between which we want to draw a line. A line is composed of two rectangles and four points as seen in Figure 38.

Figure 38: The triangle mesh of the line between m_{i-1} and m_i

At first, we calculate the difference δ and its normalized direction vector d .

$$\delta := pos(m_i) - pos(m_{i-1}), \quad d := \frac{\delta}{\|\delta\|}$$

Furthermore, we can calculate the clockwise cw and counter-clockwise orthogonal ccw vector as

$$\begin{aligned} cw([x, y]^T) &= [y, -x]^T \\ ccw([x, y]^T) &= [-y, x]^T \end{aligned}$$

Therefore, we can calculate the points A, B, C, D seen in figure 38 as

$$\begin{aligned} A &:= \text{pos}(m_{i-1}) + cw(d) \cdot t \\ B &:= \text{pos}(m_{i-1}) + ccw(d) \cdot t \\ C &:= B + \delta \\ D &:= A + \delta \end{aligned}$$

where t is the relative thickness set as a Prefabattribute.

Gauge Plot The gauge plot was mainly provided by the “Radial 180” fill method provided by Unity [141]. A configurable, fixed (i.e. discrete) color gradient was used in order to show different colors based on the value plotted.

5.3 Live Metrics

As mentioned in the methodology, the VR digital twin was used to visualize the live metrics of the MDC part of the SCC HPC Cluster using the Grafana data source proxy API [132]. A fully event-driven and concurrent polling architecture was designed, bypassing the limitations of Unity’s single threaded scripting API. This architecture is internally queries using InfluxQL, one of the query languages used by InfluxDB. The implemented architecture works with both mocked offline data and real live data provided by the production monitoring stack described in the background section.

In this section, the technical details will be explored. Firstly, the design of the InfluxQL queries using Grafana’s Frontend will be focused on. Next, the singleton data structure providing the live data through an asynchronous event system will be introduced. After that, the actual fetching and parsing of the metrics provided by Grafana will be examined in more detail. Subsequently, the event subscribers’ processing of the newly polled data will be explored. Lastly, the workings of the offline mode and the integration of mocked data into the event workflow will be examined.

5.3.1 Query Design

As previously mentioned in the methodology, the use of Grafana as a proxy greatly enhanced the workflow in designing the InfluxQL queries. It allowed for an easy visual representation of the data corresponding to the query, resulting in faster query iterations. Additionally, the use of Grafana’s own visual query builder decreased the error rate of incorrect InfluxDB queries. In this subsection, the process of using Grafana’s frontend to create the queries used in the VR software will be shown. Note that all queries used and their example responses can be found in the appendix.

Here are the steps required for creating the queries:

1. First, a new Grafana Dashboard has to be created. In this Dashboard, multiple Panels can be created, each corresponding to one set of data and nodes to be visualized. This has multiple advantages:
 - Multiple plot and chart types, such as graph, gauge, or pie charts, can be used to better understand the data.

- The visual query builder can be used to create standardized InfluxQL queries, which also shows all valid values, resulting in a faster workflow and fewer error-prone queries.

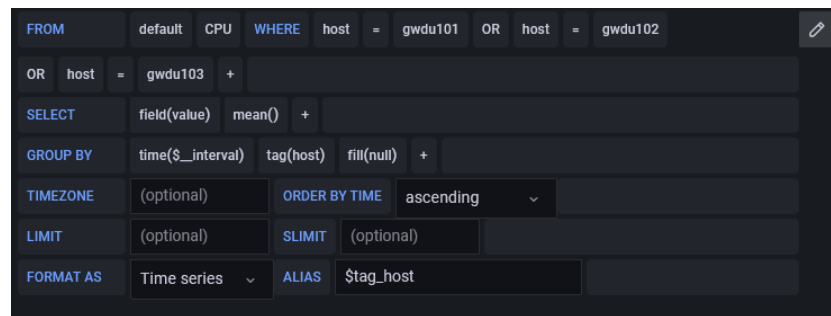


Figure 39: An example of a Query solely built with Grafana’s InfluxQL query builder

- The data can be viewed on different time frames. It can also be live updated in a regular interval, which allows to see whether the current query is a ideal for the data provided.
2. The API calls can then be extracted from the Grafana Frontend. In order to understand the idea behind this workflow, which is also mentioned in Grafanas own documentation, a small introduction into the concept of website frontends is required:

At the beginning of the internet, or the so-called Web 1.0, the content of the web was static. This means that, once the file was fetched from the webserver and the HTML and CSS were rendered by the local webbrowser, the content did not change anymore. This worked well for an information- and documentation-centered internet. Interactive elements were already possible, using either Macromedia Flash¹³, Java-Applets or Javascript, although they were only sparingly used.

Furthermore, when the user wanted to see another page, their browser had to request a new HTML file, completely fetch all embedded HTML and CSS components and rerender the whole website.

After that, people started to create more interactive websites, resulting in whole applications being ported to the browser. They heavily relied on Javascript, which was later renamed to ECMAScript, to provide the dynamic functionality.

This resulted in the concept of partitioning web applications into frontend and backend. The frontend is the part of the code that runs in the user’s browser while the backend is the code that runs on the server serving the data. After the advent of full featured web frameworks such as React, Vue or Angular, this became the standard way of creating new web applications.

In the case of Grafana, although slightly simplified, it works as follows:

¹³Later Adobe

- First, after the user requests the website, the Grafana server responds with an empty HTML struct containing links to the frontend Javascript code.
- After fetching the frontend code, the website layout is dynamically created and the frontend is initialized.
- When the user navigates the website, it is not necessary to refetch and rerender the whole website. It only needs the content in a semi-structured format¹⁴ in order to render it locally.
- Especially when new metric data is desired, the frontend only requests that data from the backend. It uses the queries previously written using the visual query builder. These requests can be recorded using the browsers development tools, replayed using our own authentication token, and the same data can be obtained.

The request sent for the metrics data can be seen in the networking tab in all modern browsers.

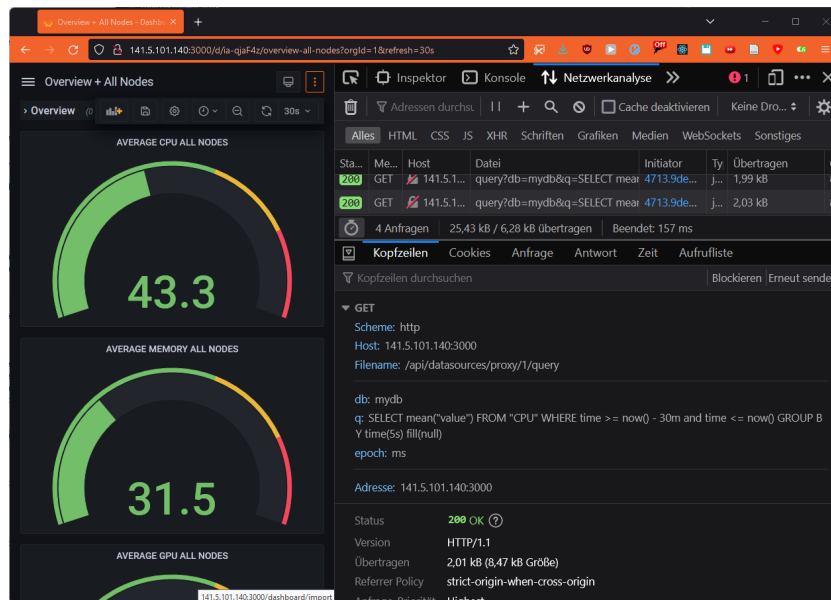


Figure 40: Viewing the Queries used by Grafana with the Firefox Developer Tools

Replaying the same request can be done either through Command Line Interface (CLI) tools such as HTTPie or cURL or more user friendly graphical alternatives such as Postman.

3. After that, the queries can be joined and further manually optimized. The full queries and truncated example responses used in the VR software can be found in the appendix.

5.3.2 Singleton Structure

The data fetching architecture is centered around the `GrafanaSingleton` class, which is, as the name suggests, a global singleton managing the fetching and parsing of the metrics

¹⁴mainly JSON

provided by a remote Grafana instance.

Here is it's UML class diagram:



Figure 41: The UML class diagram for the GrafanaSingleton

The fetched data is stored in the `cpuData`, `memData`, `gpuData`, and `storageData` dictionaries, which split the fetched data according to their metric. The keys of those dictionaries are the hostnames, which are provided by the Grafana-API as InfluxDB tags. The values are a time series of the according metric, wrapped in a `ServerData` data structure. Note that since the C# dictionary is based on a hash table, its amortized lookup complexity is $O(1)$.

The `ServerData` class can be seen as a fixed size First-In-First-Out (FIFO) queue. When constructing the queue, the maximal length must be set, which is the amount of metric data points that will be stored at a time. Each data point is a tuple of (long,

double) representing the timestamp and the scalar value representing the metric of the node at that time. After the queue is constructed, new data can be added. If the queue has too many elements, the oldest one will be removed. This queue is implemented using a double linked list, so removing the first and adding the last element are both $O(1)$. This means that this shifting operation also takes constant time.

For now, the most important parts are the **NewData** delegate and the related **NewDataArrived** event. A delegate is a type of object that can hold a reference to a method, similar to function pointers in other languages. In our case, **NewData** is a type of function that takes no parameters and returns **void**. Events are a concept for classes to notify its event subscribers. Other functions can subscribe to an event by adding their function reference of the delegate type.

This way, the **GrafanaSingleton** can poll new Data asynchronously and notify all subscribers whenever they have to update their local data visualization. Thus we can avoid costly busy polling and minimize the risk of race conditions. Furthermore, objects can unsubscribe whenever they are invisible or otherwise disabled in order to improve overall performance.

5.3.3 Fetching and Parsing Metrics

Grafana only offers access to its data through a Representational State Transfer (REST) API, which is a kind of stateless HTTP API. It should be noted that, unlike gRPC or websockets, REST APIs are unidirectional. Consequently, the server is unable to notify the client of new data. Therefore, we have to actively poll for new data.

The canonical way of repeatedly doing things in Unity is the **Update** function, invoked from Unity if the object inherits from **MonoBehaviour**. This function gets called every frame, blockingly. Thus, until the **Update** of every active object is finished, Unity won't render the next frame. As mentioned in the section about VR best practises, Oculus recommends to have at least 60 v-synced FPS [72]. Since it is impossible to fetch an API and process its response within 1/60 of a second, or 16.67 ms, the polling request can't be done synchronously. Furthermore, the data refresh rate of once a frame would be too frequent.

Unity also provides the easy to use **InvokeRepeating** method that runs the given function name repeatedly, given a specific seconds interval. Unfortunately, **InvokeRepeating** also runs in the main thread, consequently suffering from the same problem as **Update**.

In order to provide the asynchronous polling, C# coroutines were used, which are a way of writing asynchronous code. They use the **yield** keyword to return a new element. After the next call, the execution is continued after that instruction.

Awake The **GrafanaSingleton** gets initialized via the **Awake** function, which gets called by Unity at the very start of the software execution. From there, if the offline mode is not enabled, the **RealAwake** starts the **PollCoutine** routine. This coroutine runs non-blocking in its own thread.

PollCoroutine The `PollCoroutine` is an infinite loop, and for each iteration, four `async ProxyQueryRequest` Tasks are started, one for each metric (CPU, Memory, GPU, Storage). All of those run concurrently because they are not awaited one at a time, instead a `WaitUntil` is used. This works analogously to a MPI barrier or a `Promise.All` in Javascript. After that, the `ServerData` attributes are stored and the latest average value of each metric are computed and set. The first time all four REST calls have succeed, the data is formally set to be properly initialized by setting the `isInitialized` attribute to `true`. Also, after all REST calls are finished, the `NewDataArrived` event is invoked, notifying subscribers that they have to update their visualizations. Finally, execution is paused by yielding a `WaitForSeconds` object until starting the next fetching iteration. Note that no data is passed to the subscriber! All subscribers are using the data read-only, thus copying the data is avoided to improve performance.

ProxyQueryRequest Lastly, the focus will be on how the actual asynchronous request to the Grafana API is done by looking at the function of the `ProxyQueryRequest`, which takes an InfluxQL Query as a parameter. All Queries used, along with their truncated responses, can be found in the appendix. This method operates asynchronously and returns a Task, a concept also known in other languages as a Promise (in Javascript) or Future (in Rust). In order to perform the HTTP request, the `System.Net.Http.HttpClient` client is utilized, which is part of the .NET runtime. This client is the preferred modern way of making HTTP requests due to its thread safety and performance¹⁵.

The request is constructed by setting the URL to the Grafana data source proxy API endpoint, as specified in the `DATA_SOURCE_PROXY_ENDPOINT` variable. The Bearer token is then set as a HTTP header used for authentication against Grafanas user management. Furthermore, the following GET parameter are set:

- **db**: Defines the InfluxDB database which should be queried from.
- **q**: Defines the InfluxQL query used.
- **epoch**: Defines the unit in which the UNIX timestamp is returned in, always set to `ms`.

After setting the GET parameters, the asynchronous call is awaited. If the call is successful, indicated by an HTTP return code of 200 OK, the response body will contain a JSON string containing the metrics data. After that, the JSON gets deserialized and parsed into a `ServerData` struct.

JSON Deserialization The JSON string is deserialized by parsing it into a nested C# object representing the exact JSON structure. This is usually done with Unity's `JsonUtility` [143]. This is the fastest available JSON parser, since it is written in highly optimized native C++ code instead of the C# code available to scripts. Unfortunately, this library was not available for our use case; it does not support destructuring multi-dimensional JSON arrays as received by Grafana [144].

¹⁵For a taxonomy see [142]

Thus we chose to use the most popular C# based JSON parser, which is Newtonsoft's `Json.NET` [145]. This is also the next best solution from an performance perspective as shown by Jackson Dunstan [146].



Figure 42: A comparison of Unity JSON parsers [146].

5.3.4 Event Subscribers

In the last section, it was mentioned that each event subscriber is notified whenever new data arrives. The focus of this section will be on the processing of the newly fetched data by the different subscribers. To begin, we will examine the inheritance structure. After that, each class will be explored in more detail. Specifically, the solution to concurrency problem of unity's rendering being singlethreaded and not threadsafe will be discussed.

Inheritance Structure Here is the inheritance structure of all event subscribers, designed as two UML class diagrams:

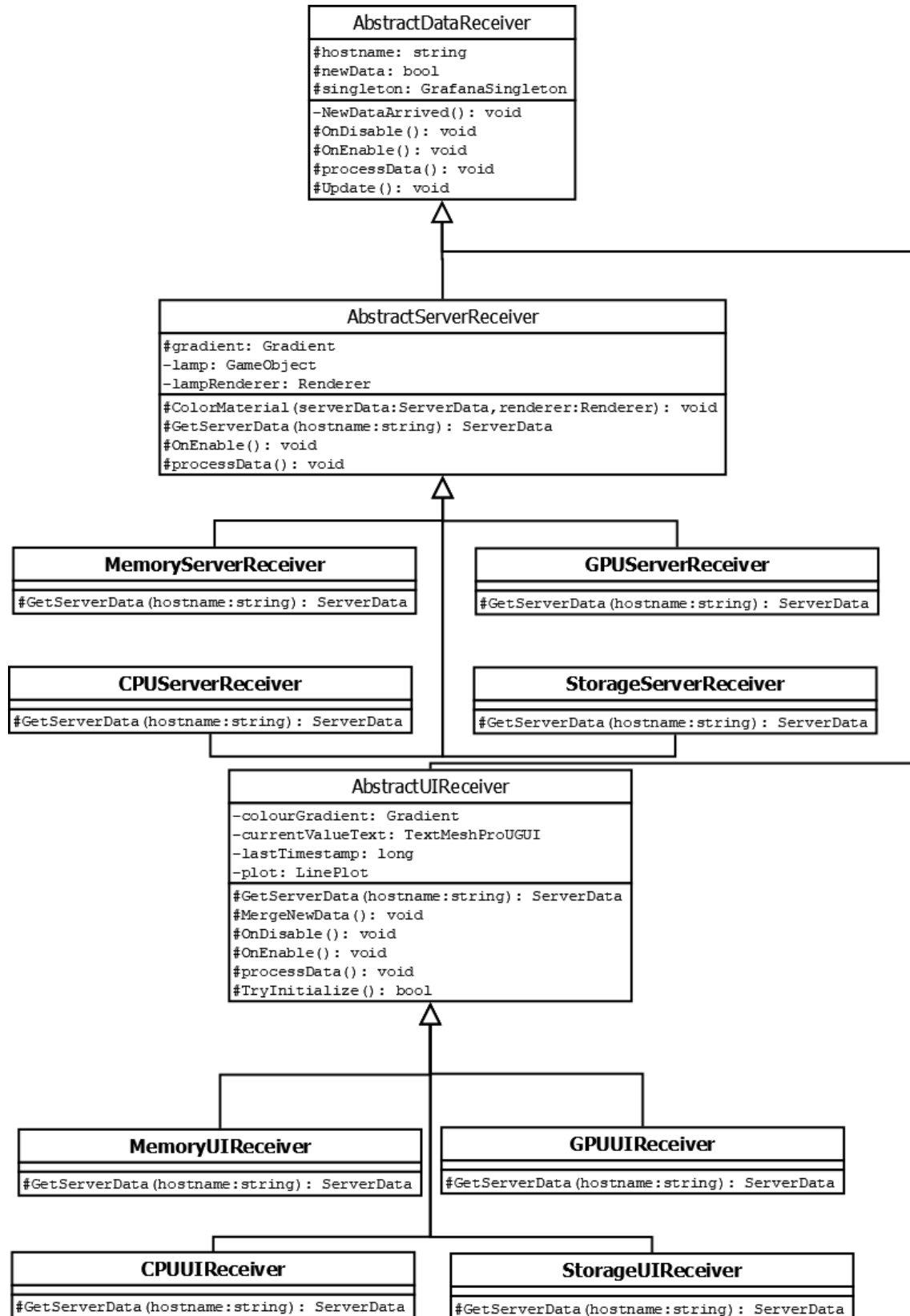


Figure 43: All event subscribers related to the UIs and server lights

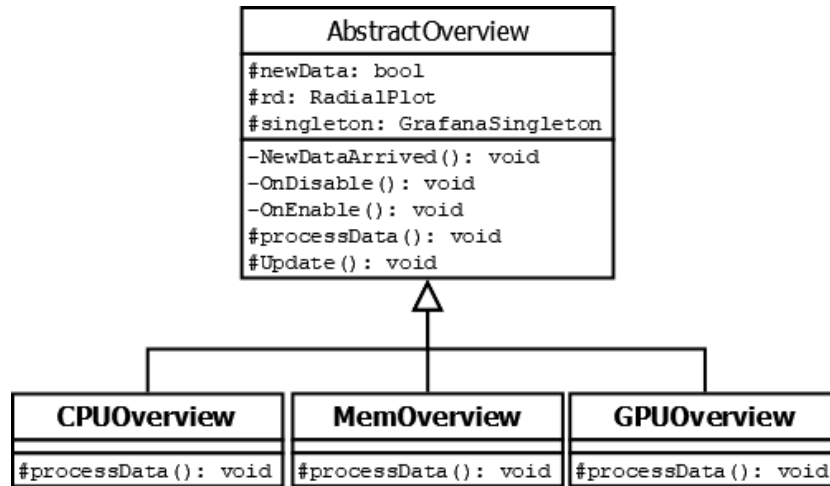


Figure 44: All event subscribers related to the initial overview

Next will be a more discussion of the single event listeners, starting with the first inheritance tree, traversing from top to bottom:

AbstractDataReceiver First, we will look at the **AbstractDataReceiver** class. The class contains the hostname of the server node as well as a reference to the **GrafanaSingleton**, which can be used in its children to extract the proper metric for the correct host. It subscribes to the **NewDataArrived** event using Unity’s **OnEnable** and unsubscribes using Unity’s **OnDisable**. In this VE, most objects are always enabled, but everything related to the UIs is enabled if and only if it is toggled visible. This optimization vastly reduces the number of subscribers.

Next we will focus on how the beforementioned concurrency problem was solved. This is the related code:

```

1 protected bool newData = true;
2
3 private void NewDataArrived()
4 {
5     newData = true;
6 }
7
8 protected void Update()
9 {
10     if (!singleton.IsInitialized)
11     {
12         return;
13     }
14     if (newData)
15     {
16         processData();
17         newData = false;
18     }
19 }
20
21 protected abstract void processData();

```

As mentioned, Unity itself is single threaded, and all rendering-related tasks must be performed in the main thread. However, as explained earlier, performing HTTP requests

synchronously in the main thread would clog it up to a point of not having enough FPS for a proper VR experience. Since everything has to be done in the main thread, some polling has to be done using the `Update` function, since this function runs on the main thread. Thus, the goal should be to minimize the time polling each frame.

At the beginning of the `Update` function, the code checks whether the singleton is formally initialized. Assuming normal network conditions, the `ServerData` structures are initialized in less than a second. Therefore, most of the time when `Update` is called, the singleton will be initialized. This is why the code branches out only in the unlikely case, improving instruction prefetching and minimizing polling time. Analogously, it is more likely that the current method invocation is not one after new data has arrived. Also, doing a boolean comparison is faster than analyzing the actual structure for changes, which further reduces the execution time of `Update`.

Every time the `GrafanaSingleton` triggers the `NewDataArrived` event, it will only set the `newData` bool of the receiver, indicating that the receiver wants to process the new data in the next frame using Unity's main thread. This solves the concurrency problem. Furthermore, it means that the concurrency problem is solved for all children of the class, as they only have to implement the `processData` method.

AbstractServerReceiver The `AbstractServerReceiver` class is an abstract class that implements coloring the material of a `GameObject` according to a set color gradient. In our VE, a green-yellow-red gradient is used to map the percentage-wise utilization to a according color. This is used for the lamps of the server racks, with each lamp having its own child of `AbstractServerReceiver`. It is worth noting that this class only processes the newest value available and does not require access to the full time series. The `AbstractServerReceiver` only exposes one abstract method, `GetServerData`, which must be implemented by all children of the class.

The other ServerReceivers The other, non-abstract `ServerReceivers` implement the `GetServerData` function to the correct `GrafanaSingleton` metric they are representing.

AbstractUIReceiver The `AbstractUIReceiver` class is an abstract class, inheriting from `AbstractDataReceiver` as seen in the UML. It is slightly more sophisticated than the `AbstractServerReceiver` since it uses the complete time series instead of just the last value. It is used for all line plots in the UIs as seen in the sections about the VE. Analogously to `AbstractServerReceiver`, it also only exposes the `GetServerData` function, which then gets implemented for each metric by one non-abstract children.

The `processData` function does the following every time new data arrives

- Merging the new data with the already plotted ones by their timestamp values.
- Rerendering the plot.
- Printing the latest percentage value.
- Coloring the UI according to the latest value, evaluated against the same gradient used for the server lamps.

Overview Receivers The overview receivers work completely equivalent to the `ServerReceiver`, but instead of changing the material color they use the previously created radial plot instead.

5.3.5 Offline Usage

If the `useDummyData` boolean in the `GrafanaSingleton` is set to `true`, the offline data compiled into the binary will be used. When the singleton initializes using the `Awake` function called by Unity, it branches to the `DummyDataAwake` function instead. In this function, the server data is filled with the values for the first 30 minutes provided by the automatically generated `DummyData` before computing the average, setting the singleton to being initialized and invoking the event. The design of the fake data and the C# code generation process can be found in the implementation chapter's user study subsection. After the initial 30 minutes of data is stored, the `InvokeRepeating` function spawns an add function every 30 seconds, simulating the real sending rate of the telegraf clients. Although `InvokeRepeating`, as mentioned before, is blocking the main thread, this is not a problem in this case as the operation of copying precomputed values is from an $O(1)$ data structure is efficient for the overhead to be negligible.

5.4 UA Study

In this section, the implementation of the User Acceptance Study will be examined. First, the generation of mocked data for the test cases described in the methodology will be discussed. Then, the implementation of the test cases will be explored, focusing on both the mocked Grafana environment and its VR digital twin.

5.4.1 Data Generation

In this section, the process of generating mock data for the UA study will be discussed. Firstly, the building blocks of the load generation will be explored. Those include two dimensional linear interpolation, function addition and clamping, as well as one-dimensional perlin noise. Secondly, it will be shown how those building blocks were used to design the different type of loads for CPU, memory, GPU and storage utilization. Finally, it will be presented how the load intensity for each node was set randomly and procedurally.

Building Blocks Since all data is represented percentagewise, the data generated lies in the $[0, 100]$ interval. The time axis is mapped to the unit interval. The generator functions were created using a functional approach of composition:

1. Simple functions that map to some part of the unit interval are used.
2. These functions are lazily layered over each other, which mathematically maps to the addition in function spaces.
3. The so-called “clamp” operation is used to map all function values within their valid interval. This ensures that the generated data remains within the required range and accurately represents the behavior of a real computing environment. It

is mathematically defined as

$$\text{clamp}(x) := \begin{cases} 100 & x \geq 100 \\ 0 & x \leq 0 \\ x & \text{otherwise} \end{cases}$$

The following basic function generators were used:

- **Constant Functions:** Constant functions are mathematically defined as

$$c_{x_0, x_1, y}(x) = \begin{cases} y & x_0 \leq x \leq x_1 \\ 0 & \text{otherwise} \end{cases}$$

- **Linear Interpolation:** We also use linear interpolation between two points, in which we evaluate the point on the convex combination of two points, and otherwise 0. Given two points $(x_0, y_0), (x_1, y_1) \in \mathbb{R}^2$, this maps to

$$L_{(x_0, y_0), (x_1, y_1)}(x) = \frac{y_0 \cdot (x_1 - x) + y_1 \cdot (x - x_0)}{x_1 - x_0}$$

- **Perlin Noise:** The `perlin-noise` python library[147] was used to generate one-dimensional perlin noise. Perlin noise is a mathematical function that generates random noise by using the values of nearby points to influence the value of each point.

Load Characteristics Different strategies were used for creating the various metric types:

- **CPU:** The CPU load generation is a multi-step process. It takes a load range (lo, hi) which define the allowed minimal and maximal utilization allowed as well as n , the number of base points and whether a lot of noise should be used. It works as follows:
 1. Generate n two-dimensional points in the domain $[0, 1) \times [lo, hi)$.
 2. Create a linear interpolation between each two neighbouring points
 3. Create the amount of perlin noise requested.
 4. Sum all linear interpolations and the perlin noise; clamp to $[0, 100]$.
- **Memory:** Typically, memory load resembles more like a step function with less variance, based on large heap allocations or new stack layers. Thus, we could not use linear interpolation to represent the load. Instead, we created the following function, taking a load range (lo, hi) and n , the number of base points as an argument:
 1. Create the set of points $S = \{0, x_1, \dots, x_n, 1\}, x_i \in [0, 1)$.
 2. For all neighbouring values $a, b \in S$ create a constant function for that interval with a random value in $[lo, hi)$.
 3. Sum all constant functions and add a bit of perlin noise.

- **GPU:** GPU metrics tend to have to most variance over time. Thus, a lot of perlin noise in order to reflect that characteristic in the mock data. Mocked GPU data consists of a one-step function of two random values, summed together with strong perlin noise.
- **Storage:** Due to its size and the short time span the storage data mostly resembles a constant function. In our mocked data, a constant function is used as well, adding a minimal amount of noise.

Procedural Load Generation By setting the $[lo, hi)$ ranges accordingly, it is then possible to define random generator function for idle, low, mid and high load for each metric. Next, we generate the node data by assigning each load intensity class a probability. For example, in the first test case, the AGT and AGQ node CPUs had a 10% probability to be idling, 20% to have low load, 60% to have mid load and 10% to be highly utilized. This, in our case, is called the weighted random load.

5.4.2 Test Cases

Next, it will be shown how the mock data was integrated into the UA studies using a standalone Python application. Specifically, we will first look at how it was streamed into a fake Grafana monitoring system. After that, we will look at the VR integration using code generation.

Mocked Grafana Instance For the user acceptance study a new grafana instance was set up. Using the GWDG cloud server [148], an Ubuntu 22.04 virtual machine was initialized and configured. Using a previously published `docker-compose` setup [149] Grafana and InfluxDB were configured.

After that, a Python application was written in order to stream mock data into the remote InfluxDB using the now deprecated `InfluxDB-Python` library [150]. It uses the previously generated data and streams it using the current timestamp. Before starting to stream, it also drops all database values and initializes the database with 30 minutes of mocked data. The code is available in the Repository.

Lastly, three Dashboards were created:¹⁶

¹⁶Only the ones with GPU notes are shown. For Test Case 2, the GPU data is omitted as it is set to zero.



Figure 45: The “Frontends and Storage” dashboard



Figure 46: The “Overview + All Nodes” dashboard



Figure 47: The “Single Node Dashboard” dashboard

All dashboards are available as a JSON export in the repository.

VR integration Since, in the user acceptance study, the starting order was alternated, all test case data had to be available in both the dashboard and VR version. To achieve this, we extended a Python application to generate a C# class containing a `Dictionary` data struct initialized with the mocked data. Additionally, for each of the four test cases, we generated a complete binary. This has a pragmatic reason:

Instead of directly embedding the Mono C# runtime, which is an alternative to Microsofts .NET core, Unity uses their own tool called IL2CPP, which generates highly optimized C++ code from C#. This code, using the Android NDK bindings to interact with the Oculus Quest 2, then gets compiled by LLVM. Unfortunately, the generated source code grows exponentially to the original one, resulting in more than 30 gigabytes of RAM usage and hour-long compile times if multiple test cases are embedded.

In the next chapter, the evaluation of the digital twin will be explored, and its results will be discussed

6 Evaluation

In Section 6.1, the participants and the structure of the user acceptance study evaluating the VR digital twin against a traditional web dashboard will be examined. In Section 6.2, the analysis methodology will be discussed, with a focus on how statistical tests were used to infer the significance of the recorded samples. In Section 6.3, the results from applying the beforementioned methodology will be presented. Finally, in Section 6.4, the results will be discussed.

6.1 Participants and Study Structure

The study was done with the following participants: The sample size for this study was $n = 20$. All participants were in at least their third undergraduate semester, with 20% being master students and 80% being bachelor students. The average age of participants was 22.75. All participants were either computer science or data science majors, with two participants completing a double major with math and biochemistry respectively. This particular sampling was a good fit for several reasons. Firstly, as previously explained in the Methodology Section, the participants were knowledgeable enough to understand the data and were able to perform basic reasoning and inferring. Additionally, none of the participants were experienced with the particular HPC cluster and metric analysis used in the study, allowing them to benefit from the spatial and dynamic features of the VE.

The reasoning behind the structure of this study is further explained in the Methodology Section. The study was structured in four parts: the introduction, pre-questionnaire, test cases, and post-questionnaire.

The introduction provided an overview of HPC systems and their batch job structure in order to provide participants with the necessary knowledge to understand and reason about the data visualized as well as an introduction to the virtual environment and its navigation. This can be found in the appendix. The pre-questionnaire collected data on students age, major, and previous experience with VR, server maintenance, and remote computing infrastructure.

Next, the four test cases were then completed in an alternated manner as mentioned in the Methodology Section. A think-out-aloud protocol was used during the test cases. This was needed since, due to the technical constraints of using a laptop without a dedicated graphics card, it was not possible to stream the application using Oculus Airlink. Lastly, the students were asked to self-assess the UX and how well the medium was suited for the data analysis task.

6.2 Analysis Methodology

Note that the data collected and its formats can be found in the appendix. The analysis focuses on the self-assessment data collected in the post-questionnaire, specifically in comparing the UX and effectiveness of the data analysis tasks performed on both platforms. We will analyze it using the following methodology, using an α of 0.05:

- To begin, the data will be visually plotted.

- Next, it will be checked whether a parametric test can be used. For this, the normality of the data has to be tested. This will be done using the Shapiro-Wilk test using `scipy.stats.shapiro` [151]. Since it is only needed to specifically test for the normal distribution, the Kolmogorov-Smirnov test is not necessary.
 - If the data is normal distributed, a parametric t-test will be used via `scipy.stats.ttest_rel` [151].
 - If the data is not normal distributed, a non-parametric Wilcoxon signed-rank test will be used via `scipy.stats.wilcoxon` [151].
- Lastly, the self-reported change in quality of the experience will be looked at.

The Jupyter notebook containing all computation as well as an Microsoft Excel table of the data can be found in the repository.

6.3 Results

The leikert scales related to the self-asesment have the following distribution:

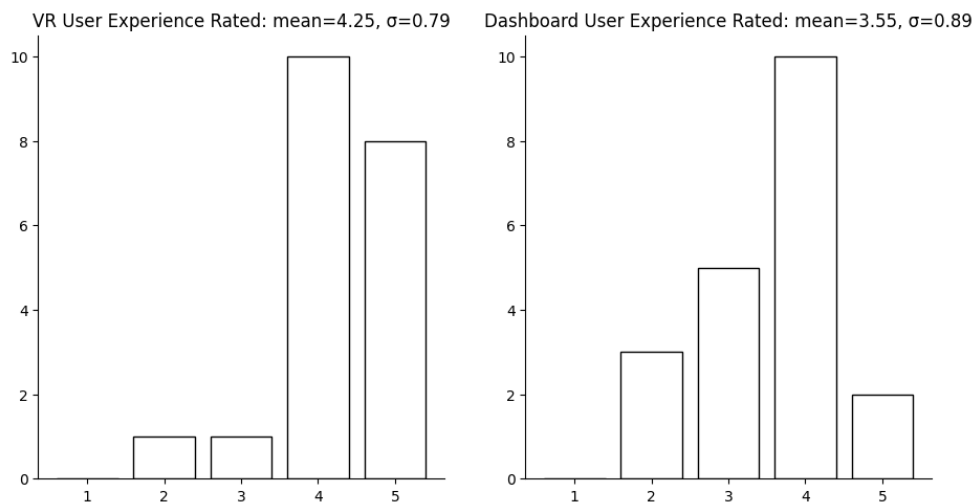


Figure 48: Distributions of the the self-reported post-questionnaire data about the VR user experience.

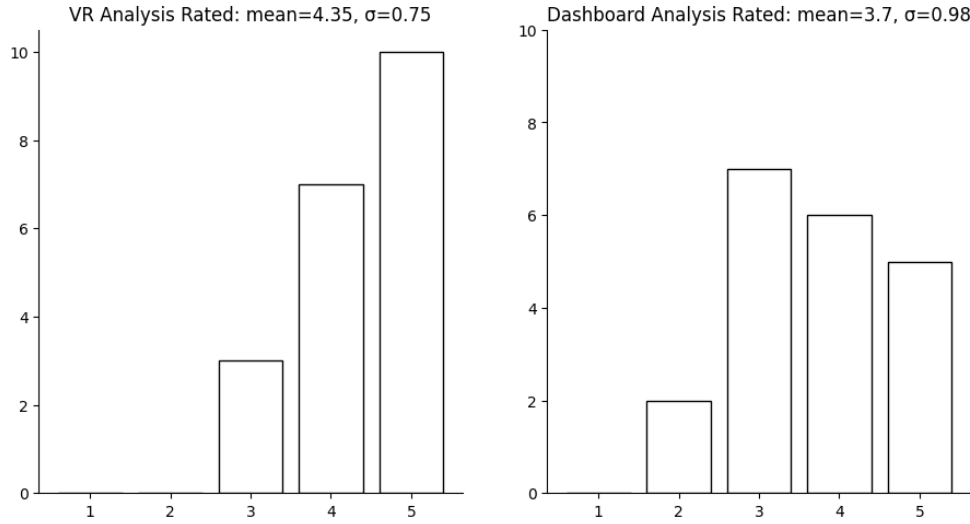


Figure 49: Distributions of the the self-reported post-questionnaire data about the VR analysis.

As one can already visually see, the recorded data is not normally distributed. Furthermore, using the Shapiro-Wilk test for normality, the following p values can be computed, confirming the highly non-normality of the sample data:

$$p_{\text{VR UX}} \approx 0.0003$$

$$p_{\text{Dashboard UX}} \approx 0.0071$$

$$p_{\text{VR Analysis}} \approx 0.0002$$

$$p_{\text{Dashboard Analysis}} \approx 0.0168$$

Thus, the Wilcoxon signed-rank test will be used, resulting in the following p and t values:

$$t_{\text{UX}} = 25.5$$

$$p_{\text{UX}} \approx 0.0222$$

$$t_{\text{Analysis}} = 9.0$$

$$p_{\text{Analysis}} \approx 0.0148$$

Those p values indicate that there is a statistically significant difference between the two groups of ratings. This means that the difference in the ratings for the web dashboard user and the VR version is unlikely to be solely coincidental. Therefore, the data suggests that the students both preferred the UX as well as the analysis capabilities of the VR version over the traditional grafana dashboards.

Lastly, we will look at VR's reported change of quality over dashboards:

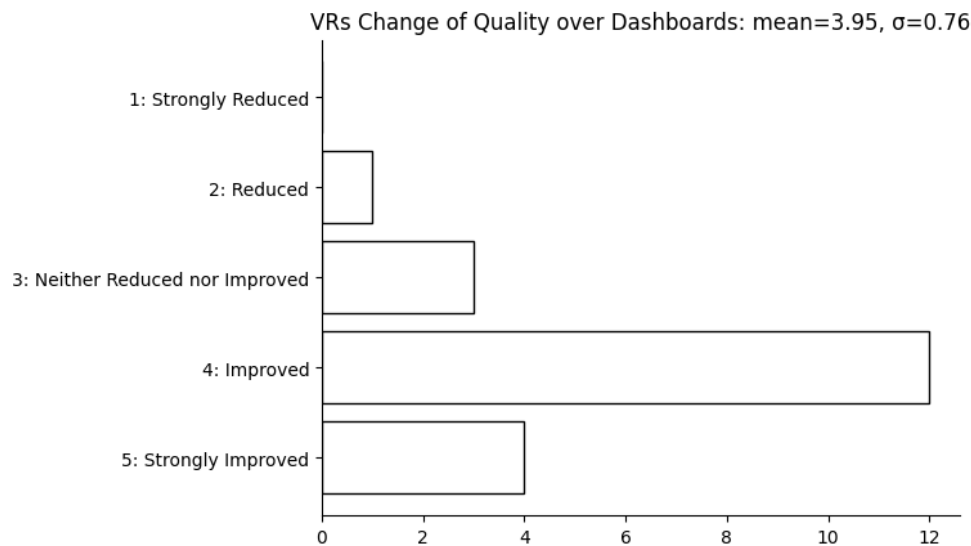


Figure 50: The distribution of the self-reported data answering the question “How did the VR version change the quality of the experience?”.

As this data is not normal distributed, it won’t be further analysed. But it can be seen that the majority of participants felt that the VR version improved the quality of the experience.

6.4 Discussion

As seen in the results it seems that the users preferred the User Experience of the VR version. This outcome was expected, given that most of them did not have a lot of previous experience with virtual reality, with only one person reporting to use their VR headset around once a month or more. Additionally, the data suggests that the participants preferred the data analysis in the VR version over the dashboards. In the think-out-loud protocol, it was often mentioned that the students liked the overview provided by the servers simply by looking at them. However, it is possible that this result is misleading for the following reason:

The spatial features of VR and immersive virtual environments provide the most benefit to the users. As mentioned twice in the post-questionnaire, the VR software most likely provides the most benefit to the user unfamiliar to the cluster. One could argue that sophisticated system administrators could prefer analyzing data using the web dashboard version due to its compactness and less usage overhead. Nonetheless, the software seems to be useful for the target audience described earlier.

7 Conclusion

Overall, this thesis showed that VR could be a beneficial addition for the visualization of data centers metrics. It revealed the potential of using a digital twin in order to leverage the spatial structure of the virtual environment, giving the user a more intuitive understanding of the data showed.

With its immersive environments and more natural interactions, virtual reality has found various applications in many diverse sectors. This thesis explored the benefits of using VR in the application field of data visualization, particularly for visualizing a HPC cluster's utilization. This has been done by developing an immersive server room digital twin using the Unity framework.

The virtual server room, spanning multiple rooms, contains various overviews aggregating the data. Furthermore, each server rack can be interacted with individually by a traditional user interface in the virtual world space. To provide those UIs, a new open source plotting library has been developed.

Moreover, an concurrent, event-based data polling architecture has been implemented to provide the live metrics of the GWDG's SCC HPC cluster. By using InfluxQL queries, it is able to poll the data using Grafana's proxy APIs. This data can then be processed by all subscribers. The architecture also allows the integration of mock data for offline usage.

In order to generate this mock data, a workflow was designed and implemented. Furthermore, this generation works completely procedural and can be adjusted to create data containing specific load characteristics. This data can then be either exported as a C# class or streamed into an InfluxDB.

Leveraging the reproducibility of virtual environments, an in-between user acceptance study with 20 participants was conducted, comparing the website-based dashboards to the VR-based digital twin. Training effects were mitigated by alternating both the starting order and the UI between the tests. The collected data is based on the participants' self-assessment.

This study revealed that the use of virtual reality provided an advantage in both the user experience ($p \approx 0.0222$, $t = 25.5$) as well as the perceived data analysis complexity ($p \approx 0.0148$, $t = 9$). Additionally, most of the participants felt that using virtual reality improved the overall quality of the experience with 16/20 participants reporting that the quality of the experience either improved or strongly improved.

Future Work There are multiple next steps in how the usage of VR for analyzing metrics could be further extended. On the one hand, more VR native interaction methods such as hand gestures could be used to further improve the immersion. On the other hand, one could create more visualizations and increase the amount of metrics visualized, for example using a Heatmap shader to visualize temperatures.

References

- [1] IDC and Statista. *Volume of data/information created, captured, copied, and consumed worldwide from 2010 to 2020, with forecasts from 2021 to 2025 (in zettabytes) [Graph]*. June 2021. URL: <https://www.statista.com/statistics/871513/worldwide-data-created/> (visited on 10/05/2022).
- [2] Tony Hey et al. *The Fourth Paradigm: Data-Intensive Scientific Discovery*. Microsoft Research, Oct. 2009. ISBN: 978-0-9825442-0-4. URL: <https://www.microsoft.com/en-us/research/publication/fourth-paradigm-data-intensive-scientific-discovery/>.
- [3] Jim Gray. “Jim Gray on eScience: A Transformed Scientific Method”. In: (Jan. 2007), pp. 1–15. URL: <http://itre.cis.upenn.edu/myl/JimGrayOnE-Science.pdf>.
- [4] HLRN. *The compute nodes of HLRN-IV phase 1 in Göttingen*. 2018. URL: https://www.hlrn.de/wp-content/uploads/2019/12/HDR_3673-3674-3675a-2000x1333.jpg (visited on 10/06/2022).
- [5] Grafana Labs. *Grafana: The open observability platform*. URL: <https://grafana.com/> (visited on 12/15/2022).
- [6] Datadog. *Datadog: Cloud Monitoring as a Service*. URL: <https://www.datadoghq.com/> (visited on 12/15/2022).
- [7] DevOps Nirvana. *Kubernetes - Nginx Ingress via Prometheus Metrics*. 2021. URL: <https://github.com/DevOps-Nirvana/Grafana-Dashboards> (visited on 10/06/2022).
- [8] SuperData Research. *Consumer virtual reality (VR) hardware and software market revenue worldwide from 2016 to 2023*. Apr. 2020. URL: <https://www.statista.com/statistics/528779/virtual-reality-market-size-worldwide/> (visited on 10/05/2022).
- [9] Unity. *Unity Real-Time Development Platform*. URL: <https://unity.com/> (visited on 12/15/2022).
- [10] InfluxData. *InfluxDB Times Series Data Platform*. URL: <https://www.influxdata.com/> (visited on 12/15/2022).
- [11] Open Source Initiative. *The MIT License*. URL: <https://opensource.org/licenses/mit-license.php> (visited on 12/15/2022).
- [12] Simon Davis, Keith Nesbitt, and Eugene Nalivaiko. “A Systematic Review of Cybersickness”. In: *Proceedings of the 2014 Conference on Interactive Entertainment*. IE2014: Interactive Entertainment 2014. Newcastle NSW Australia: ACM, Dec. 2, 2014, pp. 1–9. ISBN: 978-1-4503-2790-9. DOI: 10.1145/2677758.2677780. URL: <https://dl.acm.org/doi/10.1145/2677758.2677780> (visited on 10/03/2022).
- [13] Ivan E. Sutherland. “A Head-Mounted Three Dimensional Display”. In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. AFIPS '68 (Fall, part I). San Francisco, California: Association for Computing Machinery, 1968, pp. 757–764. ISBN: 9781450378994. DOI: 10.1145/1476589.1476686. URL: <https://doi.org/10.1145/1476589.1476686>.

- [14] Carolina Cruz-Neira, Daniel J Sandin, and Thomas A DeFanti. "Surround-Screen Projection-Based Virtual Reality: The Design and Implementation of the CAVE". In: (Jan. 1993), pp. 1–8. URL: <https://doi.org/10.1145/166117.166134>.
- [15] Dave Pape. *The Cave Automatic Virtual Environment at EVL, University of Illinois at Chicago*. 2001. URL: https://commons.wikimedia.org/wiki/File:CAVE_Crayoland.jpg (visited on 10/06/2022).
- [16] Scott Isabelle et al. "Defense applications of the CAVE (CAVE automatic virtual environment)". In: (July 1997), p. 14. DOI: 10.1117/12.277034.
- [17] Kai-Uwe Doer, Jens Schiefel, and W. Kubbat. "Virtual Cockpit Simulation for Pilot Training". In: (Mar. 2001), p. 8. URL: <https://apps.dtic.mil/sti/citations/ADP010789>.
- [18] Business Insider. *Microsoft's Kinect Is Being Used To Help Guard The Korean DMZ*. Feb. 2014. URL: <https://www.businessinsider.com/the-kinect-is-helping-monitor-the-dmz-2014-2> (visited on 10/06/2022).
- [19] Oculus. *Oculus Rift: Step Into the Game*. URL: <https://www.kickstarter.com/projects/1523379957/oculus-rift-step-into-the-game> (visited on 12/15/2022).
- [20] Guanjie Zhao et al. "The comparison of teaching efficiency between virtual reality and traditional education in medical education: a systematic review and meta-analysis". In: *Annals of Translational Medicine* 9.3 (Feb. 2021), p. 252. ISSN: 2305-5839. DOI: 10.21037/atm-20-2785. URL: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7940910/> (visited on 10/03/2022).
- [21] Taylor Gilmore et al. *Data Center of the Future: VR Construction Design Review*. June 2022. URL: https://www.youtube.com/watch?v=_E4JSRrT_D4 (visited on 10/06/2022).
- [22] US Food and Drug Administration. *FDA Authorizes Marketing of Virtual Reality System for Chronic Pain Reduction*. Nov. 2021. URL: <https://www.fda.gov/news-events/press-announcements/fda-authorizes-marketing-virtual-reality-system-chronic-pain-reduction> (visited on 10/06/2022).
- [23] PwC. *Virtual reality (VR) gaming revenue worldwide from 2017 to 2024*. Sept. 2020. URL: <https://www.statista.com/statistics/499714/global-virtual-reality-gaming-sales-revenue/> (visited on 10/06/2022).
- [24] Arzu Coltekin et al. "Extended Reality in Spatial Sciences: A Review of Research Challenges and Future Directions". In: *International Journal of Geo-Information* 9 (July 2020). DOI: 10.3390/ijgi9070439.
- [25] Google VR. *Get Cardboard*. URL: <https://arvr.google.com/cardboard/get-cardboard/> (visited on 12/15/2022).
- [26] GoogleVR. *Google Cardboard Manufacturers Kit*. URL: https://arvr.google.com/cardboard/pdfs/gc_manufacturers_kit.zip (visited on 10/06/2022).
- [27] Google. *Google Cardboard*. URL: <https://apps.apple.com/de/app/google-cardboard/id987962261> (visited on 12/15/2022).
- [28] Jeffrey Chen. *Open sourcing Google Cardboard*. Nov. 2019. URL: <https://developers.googleblog.com/2019/11/open-sourcing-google-cardboard.html> (visited on 10/06/2022).

- [29] GoogleVR. *Cardboard SDK*. URL: <https://github.com/googlevr/cardboard> (visited on 10/06/2022).
- [30] GoogleVR. *Google Cardboard XR Plugin for Unity*. URL: <https://github.com/googlevr/cardboard-xr-plugin> (visited on 10/06/2022).
- [31] othree. *An assembled Google Cardboard VR mount*. June 2014. URL: https://commons.wikimedia.org/wiki/File:Assembled_Google_Cardboard_VR_mount.jpg (visited on 10/06/2022).
- [32] Darrell Etherington. *An Inside Look At The Development Of Samsung's Gear VR At Oculus With CEO Brendan Iribe*. Sept. 2014. URL: <http://tcrn.ch/Wduq40> (visited on 10/06/2022).
- [33] Samsung. *Samsung Gear VR*. URL: https://en.wikipedia.org/wiki/Samsung_Gear_VR (visited on 12/15/2022).
- [34] Maurizio Pesce. *Samsung Gear VR*. Sept. 2014. URL: <https://www.flickr.com/photos/pestoverde/15247457825> (visited on 10/07/2022).
- [35] Inc. Within Unlimited. *WITHIN*. URL: <https://play.google.com/store/apps/details?id=com.shakingearthdigital.vrsecardboard> (visited on 12/15/2022).
- [36] Prashant Sharma. "Challenges with virtual reality on mobile devices". In: *ACM SIGGRAPH 2015 Talks*. SIGGRAPH '15. New York, NY, USA: Association for Computing Machinery, July 31, 2015, p. 1. ISBN: 978-1-4503-3636-9. DOI: 10.1145/2775280.2792597. URL: <https://doi.org/10.1145/2775280.2792597> (visited on 10/06/2022).
- [37] Chaim Gartenberg. *Meta's Oculus Quest 2 has shipped 10 million units, according to Qualcomm*. Nov. 2021. URL: <https://www.theverge.com/2021/11/16/22785469/meta-oculus-quest-2-10-million-units-sold-qualcomm-xr2> (visited on 10/07/2022).
- [38] Oculus-Blog. *Air Link, a Wireless Way to Play PC VR Games on Oculus Quest 2, Plus Infinite Office Updates, Support for 120 Hz on Quest 2, and More*. Apr. 2021. URL: <https://www.oculus.com/blog/introducing-oculus-air-link-a-wireless-way-to-play-pc-vr-games-on-oculus-quest-2-plus-infinite-office-updates-support-for-120-hz-on-quest-2-and-more/> (visited on 10/07/2022).
- [39] KKPCW. *Oculus Quest 2 - 2*. Oct. 2020. URL: https://commons.wikimedia.org/wiki/File:Oculus_Quest_2_-_2.jpg (visited on 10/07/2022).
- [40] Meta. *Oculus Quest 2: Our Most Advanced New All-in-one VR Headset*. URL: <https://www.meta.com/de/en/quest/products/quest-2/tech-specs/> (visited on 12/15/2022).
- [41] Meta. *Horizon Workrooms*. URL: <https://www.oculus.com/experiences/quest/2514011888645651/> (visited on 12/15/2022).
- [42] VRChat Inc. *VRChat*. URL: <https://www.oculus.com/experiences/quest/1856672347794301/> (visited on 12/15/2022).
- [43] Beat Games. *Beat Saber*. URL: <https://www.oculus.com/experiences/quest/2448060205267927/> (visited on 12/15/2022).
- [44] Ltd. Cloudhead Games. *Pistol Whip*. URL: <https://www.oculus.com/experiences/quest/2104963472963790/> (visited on 12/15/2022).

- [45] Inc. Rendeever. *Multi Brush*. URL: <https://www.oculus.com/experiences/quest/3438333449611263/> (visited on 12/15/2022).
- [46] Meta. *horizon Worrooms*. URL: <https://www.oculus.com/workrooms> (visited on 10/07/2022).
- [47] VRChat. *PRESS KIT - VR CHAT*. URL: <https://hello.vrchat.com/press> (visited on 10/07/2022).
- [48] Beat Games. *Beat Saber*. URL: https://store.steampowered.com/app/620980/Beat_Saber (visited on 10/07/2022).
- [49] Cloudhead Games ltd. *Pistol Whip*. URL: https://store.steampowered.com/app/1079800/Pistol_Whip (visited on 10/07/2022).
- [50] Valve. *Valve Index*. URL: <https://store.steampowered.com/valveindex> (visited on 12/15/2022).
- [51] Valve. *Valve Index*. URL: <https://www.valvesoftware.com/en/index> (visited on 10/07/2022).
- [52] Valve. *Half Life Alyx*. URL: https://store.steampowered.com/app/546560/HalfLife_Alyx/ (visited on 12/15/2022).
- [53] Bethesda Softworks. *Fallout 4 VR*. URL: https://store.steampowered.com/app/611660/Fallout_4_VR/ (visited on 12/15/2022).
- [54] Seth M. Feeman, Landon B. Wright, and John L. Salmon. “Exploration and evaluation of CAD modeling in virtual reality”. In: *Computer-Aided Design and Applications* 15.6 (Nov. 2, 2018), pp. 892–904. ISSN: 1686-4360. DOI: 10.1080/16864360.2018.1462570. URL: http://www.cad-journal.net/files/vol_15/Vol15No6.html (visited on 10/07/2022).
- [55] Omid Abari et al. “Cutting the cord in virtual reality”. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 2016, pp. 162–168.
- [56] Unity. *Game development terms*. URL: <https://unity.com/how-to/beginner/game-development-terms> (visited on 10/07/2022).
- [57] YoYo Games Ltd. *Gamemaker*. URL: <https://gamemaker.io/en> (visited on 12/15/2022).
- [58] tobyfox. *Undertale*. URL: <https://store.steampowered.com/app/391540/Undertale/> (visited on 12/15/2022).
- [59] Dennaton Games. *Hotline Miami*. URL: https://store.steampowered.com/app/219150/Hotline_Miami/ (visited on 12/15/2022).
- [60] Chromium. *Chroum Security: Memory Safety*. URL: <https://www.chromium.org/Home/chromium-security/memory-safety> (visited on 10/07/2022).
- [61] Ariel Manzur Juan Linietsky and contributors. *Godot Engine*. URL: <https://godotengine.org/> (visited on 12/15/2022).
- [62] Marcus Toftedahl and Henrik Engström. “A Taxonomy of Game Engines and the Tools that Drive the Industry”. In: Aug. 2019, p. 17.
- [63] Epic Games. *Unreal Engine*. URL: <https://www.unrealengine.com/en-US> (visited on 12/15/2022).

- [64] Valve. *Source 2*. URL: https://developer.valvesoftware.com/wiki/Source_2 (visited on 12/15/2022).
- [65] OpenGOAL. *OpenGOAL*. URL: <https://opengoal.dev/> (visited on 12/15/2022).
- [66] BEVY. *BEVY*. URL: <https://bevyengine.org/> (visited on 12/15/2022).
- [67] Roblox Corporation. *Roblox*. URL: <https://www.roblox.com/> (visited on 12/15/2022).
- [68] Steve Bryson. "Virtual reality in scientific visualization". In: *Communications of the ACM* 39.5 (May 1996), pp. 62–71. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/229459.229467. URL: <https://dl.acm.org/doi/10.1145/229459.229467> (visited on 10/03/2022).
- [69] Abraham G. Campbell et al. "Uses of Virtual Reality for Communication in Financial Services: A Case Study on Comparing Different Telepresence Interfaces: Virtual Reality Compared to Video Conferencing". In: *Advances in Information and Communication*. Ed. by Kohei Arai and Rahul Bhatia. Vol. 69. Series Title: Lecture Notes in Networks and Systems. Springer International Publishing, 2020, pp. 463–481. ISBN: 978-3-030-12387-1 978-3-030-12388-8. DOI: 10.1007/978-3-030-12388-8_33. URL: http://link.springer.com/10.1007/978-3-030-12388-8_33 (visited on 10/08/2022).
- [70] Kerry Davis. *HLVR Door Talk*. Oct. 2019. URL: <https://www.youtube.com/watch?v=9kzu2Y33yKM> (visited on 10/08/2022).
- [71] Joseph J. LaViola. "A discussion of cybersickness in virtual environments". In: *ACM SIGCHI Bulletin* 32.1 (Jan. 2000), pp. 47–56. ISSN: 0736-6906. DOI: 10.1145/333329.333344. URL: <https://dl.acm.org/doi/10.1145/333329.333344> (visited on 10/04/2022).
- [72] Richard Yao et al. *Oculus best practises*. 2014.
- [73] HLRN. *hlrn.de*. URL: <https://www.hlrn.de/> (visited on 12/15/2022).
- [74] HLRN. *Research and Development*. URL: <https://www.hlrn.de/about-us/researchdevelopment/?lang=en> (visited on 10/07/2022).
- [75] HLRN. *HLRN-IV System*. URL: <https://www.hlrn.de/supercomputer-e/hlrn-iv-system/?lang=en> (visited on 12/15/2022).
- [76] SchedMD. *Slurm Workload Manager*. URL: <https://slurm.schedmd.com/> (visited on 12/15/2022).
- [77] InfluxData. *Telegraf*. URL: <https://www.influxdata.com/time-series-platform/telegraf/> (visited on 12/15/2022).
- [78] Prometheus Authors. *Prometheus*. URL: <https://prometheus.io/> (visited on 12/15/2022).
- [79] elastic. *Kibana*. URL: <https://www.elastic.co/de/kibana/> (visited on 12/15/2022).
- [80] elastic. *Elasticsearch*. URL: <https://www.elastic.co/de/elasticsearch/> (visited on 12/15/2022).
- [81] Andrea Vázquez-Ingelmo, Francisco J. García-Peñalvo, and Roberto Therón. "Information Dashboards and Tailoring Capabilities - A Systematic Literature Review". In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 109673–109688. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2933472.

- [82] Alper Sarikaya et al. “What Do We Talk About When We Talk About Dashboards?” In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 682–692. ISSN: 1941-0506. DOI: 10.1109/TVCG.2018.2864903.
- [83] Jessica Hullman, Eytan Adar, and Priti Shah. “The impact of social information on visual judgments”. In: *Proceedings of the SIGCHI conference on human factors in computing systems*. 2011, pp. 1461–1470.
- [84] Yea-Seul Kim, Katharina Reinecke, and Jessica Hullman. “Data through others’ eyes: The impact of visualizing others’ expectations on visualization interpretation”. In: *IEEE transactions on visualization and computer graphics* 24.1 (2017), pp. 760–769.
- [85] Lisa Pappas and Lisa Whitman. “Riding the technology wave: Effective dashboard data visualization”. In: *Symposium on Human Interface*. Springer. 2011, pp. 249–258.
- [86] European Environment Agency. *Dos and don’ts of data visualisation*. URL: <https://www.eea.europa.eu/data-and-maps/daviz/learn-more/chart-dos-and-donts> (visited on 10/10/2022).
- [87] Colblindor. *Coblis — Color Blindness Simulator*. URL: <https://www.color-blindness.com/coblis-color-blindness-simulator/> (visited on 10/10/2022).
- [88] Holoviz. *Colorcet: Collection of perceptually uniform colormaps*. URL: <https://colorcet.holoviz.org/> (visited on 10/09/2022).
- [89] Edward Tufte. *The Visual Display of Quantitative Information*. 1983.
- [90] Patrick Millais, Simon L. Jones, and Ryan Kelly. “Exploring Data in Virtual Reality: Comparisons with 2D Data Visualizations”. In: *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*. CHI ’18: CHI Conference on Human Factors in Computing Systems. Montreal QC Canada: ACM, Apr. 20, 2018, pp. 1–6. ISBN: 978-1-4503-5621-3. DOI: 10.1145/3170427.3188537. URL: <https://dl.acm.org/doi/10.1145/3170427.3188537> (visited on 10/03/2022).
- [91] Benjamin JH Andersen et al. “Immersion or diversion: Does virtual reality make data visualisation more effective?” In: *2019 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE. 2019, pp. 1–7.
- [92] Theophilus Teo et al. “Data fragment: Virtual reality for viewing and querying large image sets”. In: *2017 IEEE Virtual Reality (VR)*. IEEE. 2017, pp. 327–328.
- [93] Leaseweb. *Data center 360 VR tour - LeaseWeb*. URL: <https://www.youtube.com/watch?v=sS7UxiCVqrg> (visited on 10/10/2022).
- [94] IronMountain. *VA-1 360 VR Data Center Tour*. URL: <https://www.ironmountain.com/resources/multimedia/v/va-1-360-vr-data-center-tour> (visited on 10/10/2022).
- [95] Hetzner. *Virtual tour of Hetzner Online Data Center Park*. URL: <https://www.hetzner.com/unternehmen/360-tour/> (visited on 10/10/2022).
- [96] Teatime Research. *Teatime Research*. URL: <https://teatimeresearch.com/> (visited on 12/15/2022).

- [97] Jayant Thatte et al. "Stacked omnistereo for virtual reality with six degrees of freedom". In: *2017 IEEE Visual Communications and Image Processing (VCIP)*. IEEE. 2017, pp. 1–4.
- [98] Teatime. *Green Mountain VR*. URL: <https://teatimeresearch.com/fwportfolio/green-mountain/> (visited on 10/10/2022).
- [99] Axonom. *Powertrak VR Product Configurator*. URL: <https://www.axonom.com/vr-product-configurator> (visited on 10/10/2022).
- [100] Axonom. *Virtual Reality Data Center Rack Configuration Experience*. URL: https://www.youtube.com/watch?v=_gsh1YF6mFY (visited on 10/10/2022).
- [101] Hypercane Studios. *VR Data Center Simulator*. URL: <https://hypercanestudios.com/ptta-portfolio/vr-data-center-simulator> (visited on 10/10/2022).
- [102] dtm. *Durchblick im RZ - dtm group als erster mit VR im RZ-Planungsprozess*. URL: <https://www.openpr.de/news/1038615/Durchblick-im-RZ-dtm-group-als-erster-mit-VR-im-RZ-Planungsprozess.html> (visited on 10/10/2022).
- [103] Alan Renouf. *VMworld 2017 Europe - General Session Day 1*. URL: <https://www.youtube.com/watch?v=jOpsBC1EuNs&t=2859s> (visited on 10/10/2022).
- [104] Alan et al. Renouf. *vr-dc-ex*. URL: <https://github.com/vmware-archive/vr-dc-ex> (visited on 10/10/2022).
- [105] The Irregular Corporation. *PC Building Simulator*. URL: https://store.steampowered.com/app/621060/PC_Building_Simulator/ (visited on 10/10/2022).
- [106] Bethesda Softworks. *The Elder Scrolls V: Skyrim VR*. URL: https://store.steampowered.com/app/611670/The_Elder_Scrolls_V_Skyrim_VR/ (visited on 12/15/2022).
- [107] Wikipadia. *Creation Engine*. URL: https://en.wikipedia.org/wiki/Creation_Engine (visited on 12/15/2022).
- [108] Wikipedia. *Rockstar Advanced Game Engine*. URL: https://en.wikipedia.org/wiki/Rockstar_Advanced_Game_Engine (visited on 12/15/2022).
- [109] Alissa McAloon. *No plans for a full Source 2 SDK, and other tidbits from Valve's Half-Life: Alyx AMA*. URL: <https://www.gamedeveloper.com/design/no-plans-for-a-full-source-2-sdk-and-other-tidbits-from-valve-s-i-half-life-alyx-i-ama> (visited on 10/18/2022).
- [110] Meta. *OpenXR Mobile SDK*. URL: <https://developer.oculus.com/documentation/native/android/mobile-intro/> (visited on 12/15/2022).
- [111] Meta. *PC SDK*. URL: <https://developer.oculus.com/documentation/native/pc/pcsdk-intro/> (visited on 12/15/2022).
- [112] Meta. *Introduction to Meta Quest Browser*. URL: <https://developer.oculus.com/documentation/web/browser-intro/> (visited on 12/15/2022).
- [113] Stackoverflow. *2022 Developer Survey*. URL: <https://survey.stackoverflow.co/2022> (visited on 10/19/2022).
- [114] Pieter. *What's the difference between "STL" and "C++ Standard Library"?* URL: <https://stackoverflow.com/q/5205491/9958281> (visited on 10/19/2022).
- [115] MaxL. *Why doesn't UE utilize STL containers?* URL: <https://forums.unrealengine.com/t/why-doesnt-ue-utilize-stl-containers/34551> (visited on 10/19/2022).

- [116] Reddit. *Reddit*. URL: <https://www.reddit.com> (visited on 12/15/2022).
- [117] Reddit. *Unity 3D - News, Showcase, Help, and Discussion*. URL: <https://www.reddit.com/r/Unity3D/> (visited on 12/15/2022).
- [118] Reddit. *Unreal Engine*. URL: <https://www.reddit.com/r/unrealengine/> (visited on 12/15/2022).
- [119] udemy. *About Us*. URL: <https://about.udemy.com/> (visited on 10/19/2022).
- [120] justkevin. *Engines used in the most popular Steam games of 2020*. URL: https://www.reddit.com/r/gamedev/comments/os0idx/engines_used_in_the_most_popular_steam_games_of/ (visited on 10/19/2022).
- [121] itch.io. *Most used Engines*. URL: <https://itch.io/game-development/engines/most-projects> (visited on 10/19/2022).
- [122] BitSplash Interactive. *Graph and Chart - Scientific*. URL: <https://assetstore.unity.com/packages/tools/gui/graph-and-chart-scientific-195222> (visited on 12/15/2022).
- [123] Happy Pixels. *Awesome Charts and Graphs*. URL: <https://assetstore.unity.com/packages/tools/gui/awesome-charts-and-graphs-138153> (visited on 12/15/2022).
- [124] Unity. *Asset Store Terms of Service and EULA*. URL: <https://unity.com/legal/as-terms> (visited on 10/19/2022).
- [125] Lars Quentin. *Plotty: 2D plots in Unity3D!* URL: <https://github.com/lquenti/PlottyUnity> (visited on 10/19/2022).
- [126] Code Monkey. *Create a Graph - Unity Tutorial*. URL: <https://www.youtube.com/playlist?list=PLzDRvYVwl53v5ur4GluoabyckImZz3TVQ> (visited on 10/19/2022).
- [127] Saticmotion. *Generating UI meshes in Unity*. URL: <https://saticmotion.github.io/Blog/2018/06/26/Generating-UI-Meshes-in-Unity.html> (visited on 10/19/2022).
- [128] Martín Pane. *Graphy - Ultimate FPS Counter - Stats Monitor & Debugger (Unity)*. URL: <https://github.com/Tayx94/graphy> (visited on 10/19/2022).
- [129] Unity. *ShaderLab*. URL: <https://docs.unity3d.com/Manual/SL-Reference.html> (visited on 12/15/2022).
- [130] Microsoft. *High-level shader language (HLSL)*. URL: <https://learn.microsoft.com/en-us/windows/win32/direct3dhls/dx-graphics-hls> (visited on 12/15/2022).
- [131] Khronos. *Core Language (GLSL)*. URL: [https://www.khronos.org/opengl/wiki/Core_Language_\(GLSL\)](https://www.khronos.org/opengl/wiki/Core_Language_(GLSL)) (visited on 12/15/2022).
- [132] Grafana Labs. *Data source API*. URL: https://grafana.com/docs/grafana/latest/developers/http_api/data_source/ (visited on 12/15/2022).
- [133] GWDG. *Cisco AnyConnect (Windows)*. URL: https://docs.gwdg.de/doku.php?id=en:services:network_services:vpn:anyconnect (visited on 12/15/2022).
- [134] Edge Security. *Wireguard*. URL: <https://www.wireguard.com/> (visited on 12/15/2022).

- [135] GWDG. *SCC - GWDG - IT in der Wissenschaft*. URL: <https://www.gwdg.de/hpc-on-campus/scc> (visited on 12/12/2022).
- [136] Meta. *Oculus Hand Models*. URL: https://developer.oculus.com/downloads/package/oculus-hand-models/?locale=de_DE (visited on 12/15/2022).
- [137] Valem Tutorials. *How to Make a VR Game in Unity 2022 - PART 2 - INPUT and HAND PRESENCE*. URL: <https://www.youtube.com/watch?v=8PCNNro7Rt0> (visited on 12/15/2022).
- [138] ipoly3d. *LowPoly Server Room Props*. URL: <https://ipoly3d.com/assets/lowpoly-server-room-props/> (visited on 12/15/2022).
- [139] Unity. *MonoBehaviour*. URL: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html> (visited on 12/15/2022).
- [140] Unity. *Graphic*. URL: <https://docs.unity3d.com/2017.3/Documentation/ScriptReference/UI.Graphic.html> (visited on 12/15/2022).
- [141] Unity. *Image.FillMethod.Radial180*. URL: <https://docs.unity3d.com/2017.4/Documentation/ScriptReference/UI.Image.FillMethod.Radial180.html> (visited on 12/15/2022).
- [142] Stackoverflow community wiki. *Send HTTP POST request in .NET*. URL: <https://stackoverflow.com/questions/4015324/send-http-post-request-in-net/4015346#4015346> (visited on 12/15/2022).
- [143] Unity. *JsonUtility*. URL: <https://docs.unity3d.com/ScriptReference/JsonUtility.html> (visited on 12/15/2022).
- [144] fermmmmm. *JsonUtility fails to convert 2D arrays to json and also from json*. URL: <https://forum.unity.com/threads/jsonutility-fails-to-convert-2d-arrays-to-json-and-also-from-json.387884/> (visited on 12/15/2022).
- [145] Newtonsoft. *Json.NET*. URL: <https://www.newtonsoft.com/json> (visited on 12/15/2022).
- [146] Jackson Dunstan. *JSON Performance Benchmarks*. URL: <https://www.jacksondunstan.com/articles/3294> (visited on 12/12/2022).
- [147] Ildar SALAKHIEV. *Perlin Noise*. URL: https://github.com/salaxieb/perlin_noise (visited on 12/13/2022).
- [148] GWDG. *GWDG Cloud Server*. URL: <https://www.gwdg.de/server-services/gwdg-cloud-server> (visited on 12/15/2022).
- [149] GWDG. *InfluxBenchmarker*. URL: <https://github.com/gwdg/InfluxBenchmarker/tree/main/docker-influx-grafana> (visited on 12/15/2022).
- [150] InfluxData. *InfluxDB-Python*. URL: <https://github.com/influxdata/influxdb-python> (visited on 12/13/2022).
- [151] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

A Grafana Proxy Queries

A.1 CPU Data Query

```
1 SELECT
2   mean("load1")/max("n_cpus") * 100
3 FROM
4   "system"
5 WHERE (
6   "host"='gwdu101' OR
7   "host"='gwdu102' OR
8   "host"='gwdu103' OR
9   "host" =~ /^agt\d\d\d$/ OR
10  "host" =~ /^agq\d\d\d$/ OR
11  "host" =~ /^amp0(?:0[0-9]|1[1-6])$/
12 ) AND time >= now() - 30m
13 AND time <= now()
14 GROUP BY
15   time(30s),
16   "host"
17   fill(null)
```

A.2 CPU Data Example Response (Truncated)

```

1 {
2   "results": [
3     {
4       "statement_id": 0,
5       "series": [
6         {
7           "name": "system",
8           "tags": {
9             "host": "agq001"
10          },
11          "columns": [
12            "time",
13            "mean_max"
14          ],
15          "values": [
16            [
17              1670675640000,
18              null
19            ],
20            [
21              1670675670000,
22              0.09375
23            ],
24            ...
25          ]
26        },
27        {
28          "name": "system",
29          "tags": {
30            "host": "agq002"
31          },
32          "columns": [
33            "time",
34            "mean_max"
35          ],
36          "values": [
37            [
38              1670675640000,
39              null
40            ],
41            [
42              1670675670000,
43              17.8125
44            ],
45            ...
46          ]
47        },
48        ...
49      ]
50    }
51  ]
52 }

```

A.3 Memory Data Query

```
1 SELECT
2     mean("used_percent")
3 FROM
4     "mem"
5 WHERE (
6     "host"='gwdu101' OR
7     "host"='gwdu102' OR
8     "host"='gwdu103' OR
9     "host" =~ /^agt\d\d\d$/ OR
10    "host" =~ /^agq\d\d\d$/ OR
11    "host" =~ /^amp0(?:0[0-9]|1[1-6])$/
12 ) AND time >= now() - 30m
13 AND time <= now()
14 GROUP BY
15     time(30s),
16     "host"
17     fill(null)
```

A.4 Memory Data Example Response (Truncated)

```

1 {
2   "results": [
3     {
4       "statement_id": 0,
5       "series": [
6         {
7           "name": "mem",
8           "tags": {
9             "host": "agq001"
10          },
11          "columns": [
12            "time",
13            "mean"
14          ],
15          "values": [
16            [
17              1670676270000,
18              null
19            ],
20            [
21              1670676300000,
22              3.7886977251484337
23            ],
24            ...
25          ]
26        },
27        {
28          "name": "mem",
29          "tags": {
30            "host": "agq002"
31          },
32          "columns": [
33            "time",
34            "mean"
35          ],
36          "values": [
37            [
38              1670676270000,
39              null
40            ],
41            [
42              1670676300000,
43              8.756937081211326
44            ],
45            ...
46          ]
47        },
48        ...
49      ]
50    }
51  ]
52 }

```

A.5 GPU Data Query

```
1 SELECT
2   mean("utilization.gpu")
3 FROM
4   "nvidia_gpu"
5 WHERE (
6   "gpu_name" = 'Tesla V100-PCIE-32GB' OR
7   "gpu_name" = 'Quadro RTX 5000'
8 ) AND time >= now() - 30m
9   AND time <= now()
10 GROUP BY
11   time(30s),
12   "host"
13   fill(null)
```

A.6 GPU Data Example Response (Truncated)

```

1 {
2   "results": [
3     {
4       "statement_id": 0,
5       "series": [
6         {
7           "name": "nvidia_gpu",
8           "tags": {
9             "host": "agq001"
10          },
11          "columns": [
12            "time",
13            "mean"
14          ],
15          "values": [
16            [
17              1670676450000,
18              null
19            ],
20            [
21              1670676480000,
22              0
23            ],
24            ...
25          ]
26        },
27        {
28          "name": "nvidia_gpu",
29          "tags": {
30            "host": "agq002"
31          },
32          "columns": [
33            "time",
34            "mean"
35          ],
36          "values": [
37            [
38              1670676450000,
39              null
40            ],
41            [
42              1670676480000,
43              5
44            ],
45            ...
46          ]
47        },
48        ...
49      ]
50    }
51  ]
52 }

```


A.7 Storage Data Query

```
1 SELECT
2     last("used_percent")
3 FROM
4     "disk"
5 WHERE (
6     "host" = 'gwdu108' AND
7     "device" = 'beegfs_nodev'
8 ) AND time >= now() - 30m
9     AND time <= now()
10 GROUP BY
11     time(30s),
12     "host"
13     fill(null)
```

A.8 Storage Data Example Response (Truncated)

```
1 {
2   "results": [
3     {
4       "statement_id": 0,
5       "series": [
6         {
7           "name": "disk",
8           "tags": {
9             "host": "gwdu108"
10          },
11          "columns": [
12            "time",
13            "last"
14          ],
15          "values": [
16            [
17              1670676600000,
18              null
19            ],
20            [
21              1670676630000,
22              84.07970595904321
23            ],
24            ...
25          ]
26        }
27      ]
28    }
29  ]
30 }
```

B UA Study Template

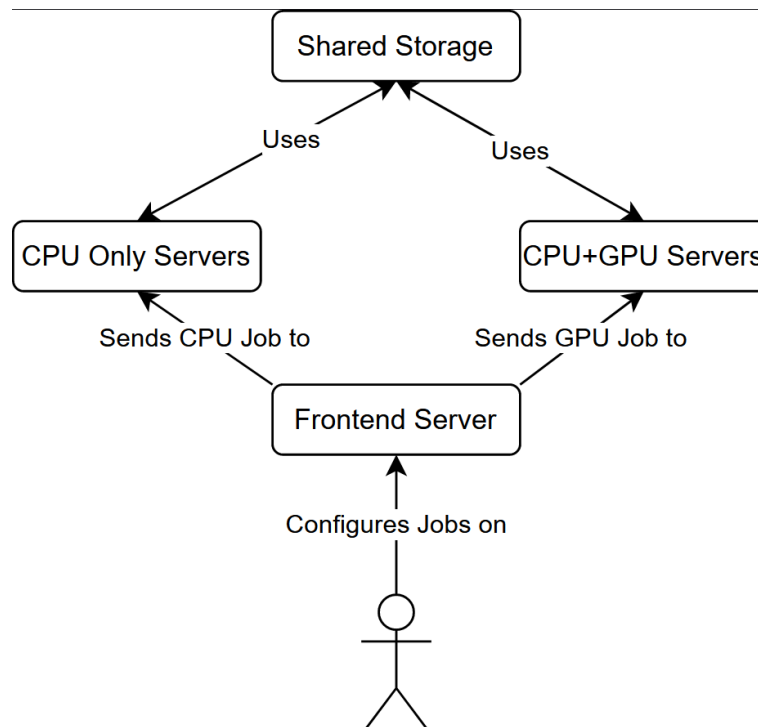
(Originally in Markdown, ported to LaTeX with the help of pandoc¹⁷)

ID: 1

Date: dd.mm.yyyy

B.1 Part 1. Topic Explanation

B.1.1 How the GWDG HPC System works



B.1.2 How to submit a job

- The User first connects to one of the so-called “Frontend-Servers” via SSH
- The Frontends are used to create the Jobs (Shell-Scripts) and compile/configure the software used for computation. They should not be used for computation since it is shared with other users that configure their jobs!
 - This is the logical equivalent to the ssh login servers at the IfI.
- Then, the User decides a “Job Queue”. By choosing a Queue, it is decided which kind of server the job gets executed on.
- In our simplified case, the user can decide between:
 - Servers without a GPU (for only computing on the CPU)
 - Servers with a GPU (for mainly computing on the GPU)

¹⁷<https://pandoc.org/>

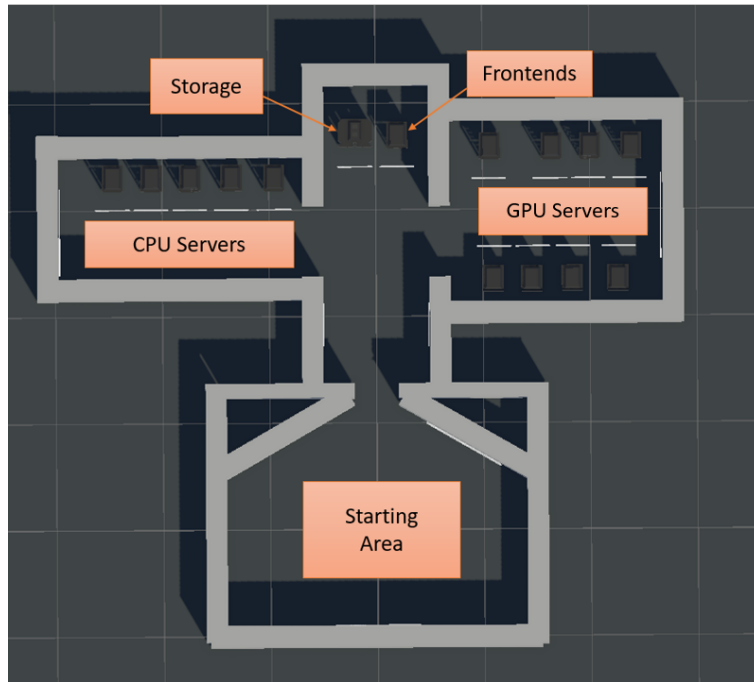
- Then, it gets put into that “First-In-First-Out” Queue. Eventually, it will be in the first place and the computation will be done.
- Afterwards, the user gets notified and get the results from the frontend server.

B.1.3 VR Environment

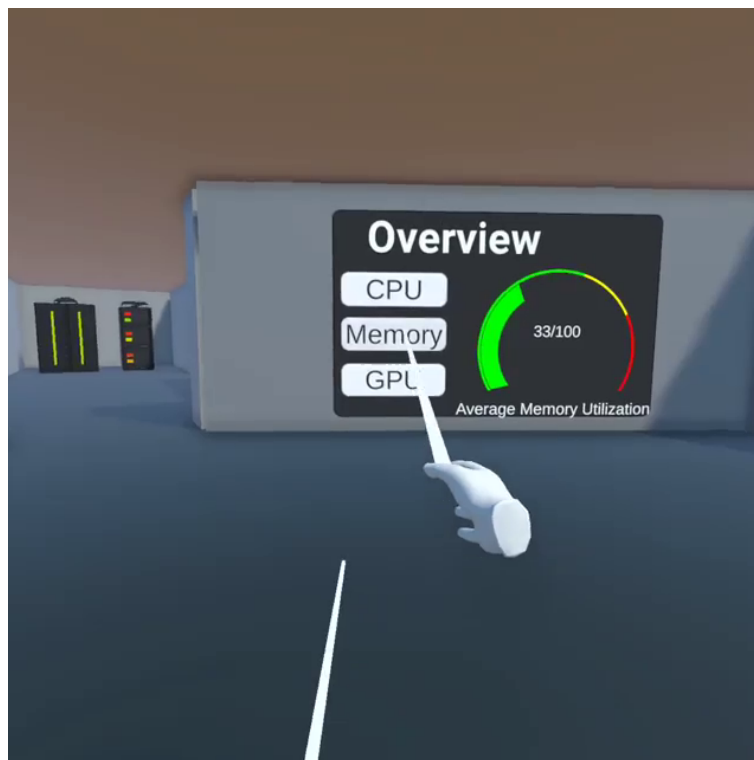


Button Bindings

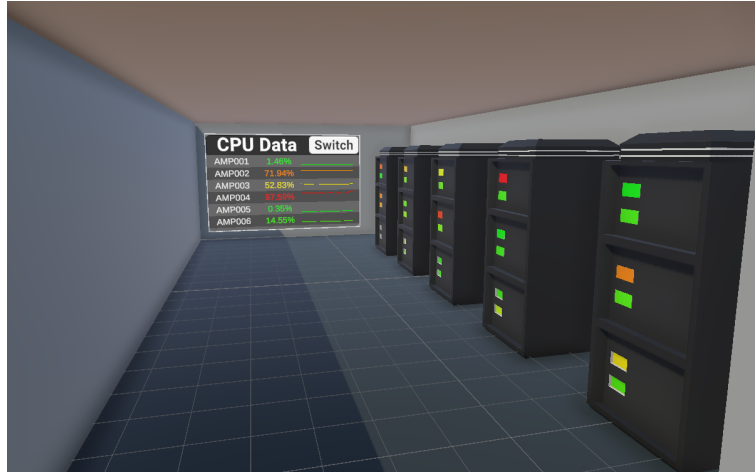
How the Environment is Structured Here is a general Overview of the Server Room:



In the starting area, you can see the total Overview, as well as the Frontend and Storage servers. The buttons can be interacted by aiming at the back button with the ray.



On the left side, one can then see the CPU room on the left side



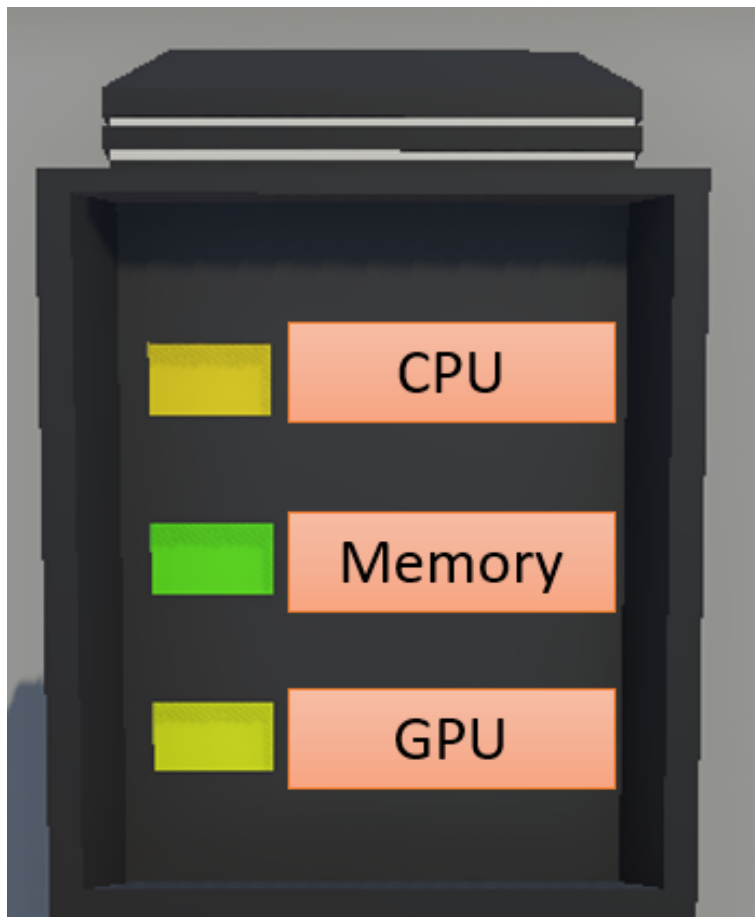
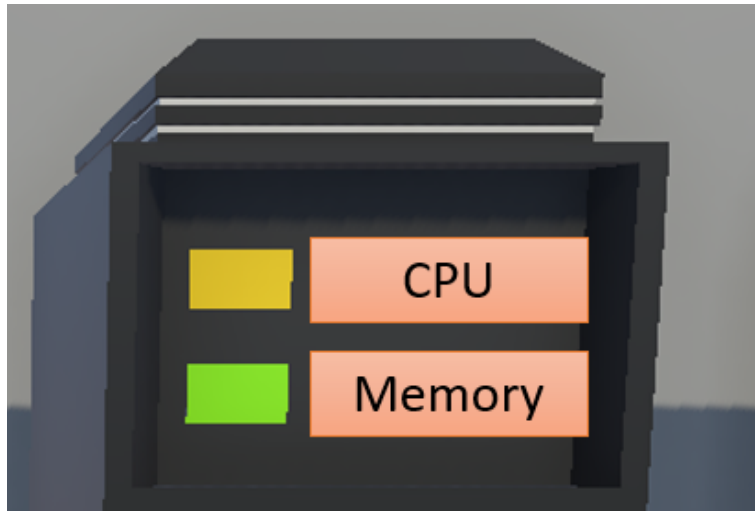
and the GPU Room on the right side



There are multiple Ways to interact with the data:

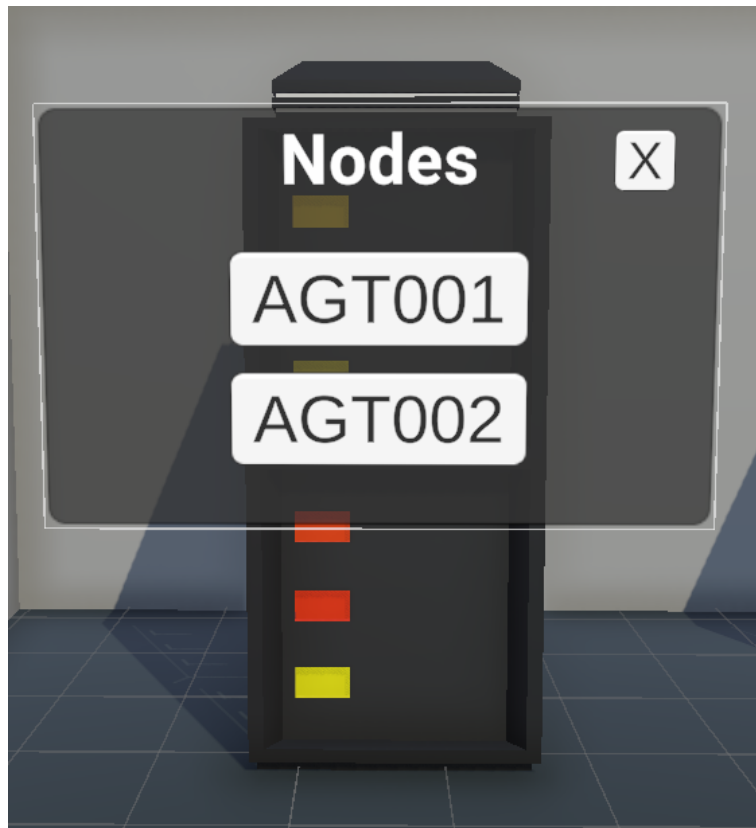
The Server

Server Lamps The server lamps indicate the current load. Servers without a GPU have two lamps, servers with a GPU have three.

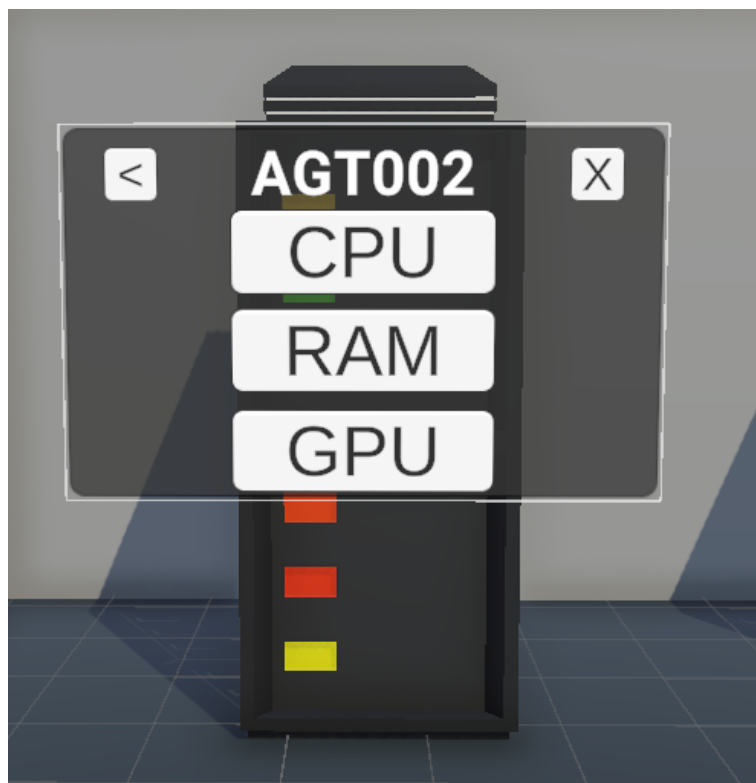


This can help for an easy overview.

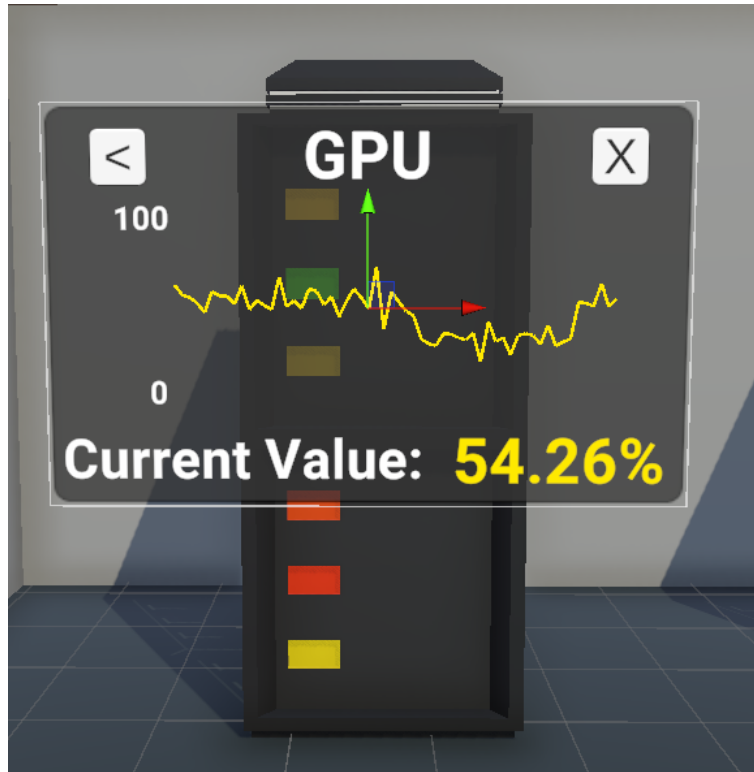
Server Menu By clicking when selecting the server one opens its metric menu. In the first menu, one can see all servers that are part of the current rack.



When selecting a specific node, one can furthermore select which metric one wants to see.

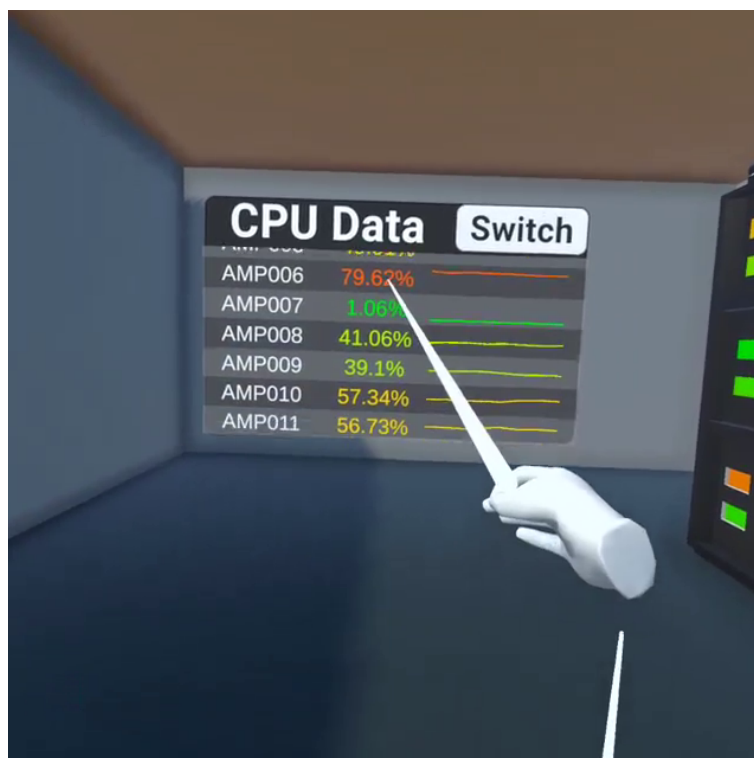


Then one sees a 30 minute overview of the previous utilization.



Note that the UI rotates towards the users head in order to be usable from every angle.

The other Overviews Besides the Server interaction and the general overview both the CPU and GPU servers have an tabular overview that shows the current metric as well as a plot of the last few minutes. The current shown metric can be toggled in the top right corner. It can be furthermore interacted by scrolling with the ray.



B.2 Part 2. Pre-Questionnaire

- Age:
- Major:
- Highest Level of Education:
 - ☐ Below High School equivalent
 - ☐ Abitur/Fachhochschulreise/High School Equivalent
 - ☐ Bachelors
 - ☐ Masters
 - ☐ PhD
- How often do you manage or maintain servers?
 - ☐ Never
 - ☐ Highly Irregular
 - ☐ Around Once a Month
 - ☐ Around Once a Week
 - ☐ Daily or Every Few Days
- How often do you use remote computing infrastructure? Both HPC and Google Colab / Hosted Jupyter count.
 - ☐ Never
 - ☐ Highly Irregular
 - ☐ Around Once a Month
 - ☐ Around Once a Week
 - ☐ Daily or Every Few Days
- How often do you use VR?
 - ☐ Never
 - ☐ Highly Irregular
 - ☐ Around Once a Month
 - ☐ Around Once a Week
 - ☐ Daily or Every Few Days
- ☐ Do you own a VR headset or have you previously owned one?
 - If Yes, which type of VR headset?
 - ☐ Smartphone-based (such as Google Cardboard)
 - ☐ Standalone Devices (such as Oculus Quest)
 - ☐ PCVR-Devices (such as HTC-Vive or Steam Index)

B.3 Part 3. Test Cases

If `id%2==1`, it starts with the Dashboardversion.

B.3.1 Test Case 1: Are the servers used optimally?

Task: Find out whether the servers could be used more optimally.

Hint: Remember how the assignment workflow works.

B.3.2 Test Case 2: How does the scheduler work?

Task: Find out how the batch tasks get scheduled by looking at the server utilization

B.3.3 Test Case 3: Long Term Decision Making

Task:

Your department got some funds and decides on whether to upgrade the current computing hardware. The hardware should just get upgraded if it is mostly fully utilized.

Find out if there is a need to upgrade the current hardware. If there is, find out which type of servers are most needed.

B.4 Part 4. Post Questionnaire

- Rate the first dashboard user experience from 1 to 5

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

- Rate the first VR user experience from 1 to 5

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4
- ☐ 5

- How easy was it to analyze the data in the dashboards

- ☐ 1
- ☐ 2
- ☐ 3
- ☐ 4

☐ 5

- How easy was it to analyze the data in VR

☐ 1

☐ 2

☐ 3

☐ 4

☐ 5

- How did the VR version change the quality of the experience?

☐ Strongly Reduced

☐ Reduced

☐ Neither Reduced nor Improved

☐ Improved

☐ Strongly Improved

- What did you like the most about the VR version

<Text here>

- What did you like the least about the VR version

<Text here>

C Mocked Data used in UA Study

C.1 Test Case 1: Are the servers used optimally?

Frontend Servers

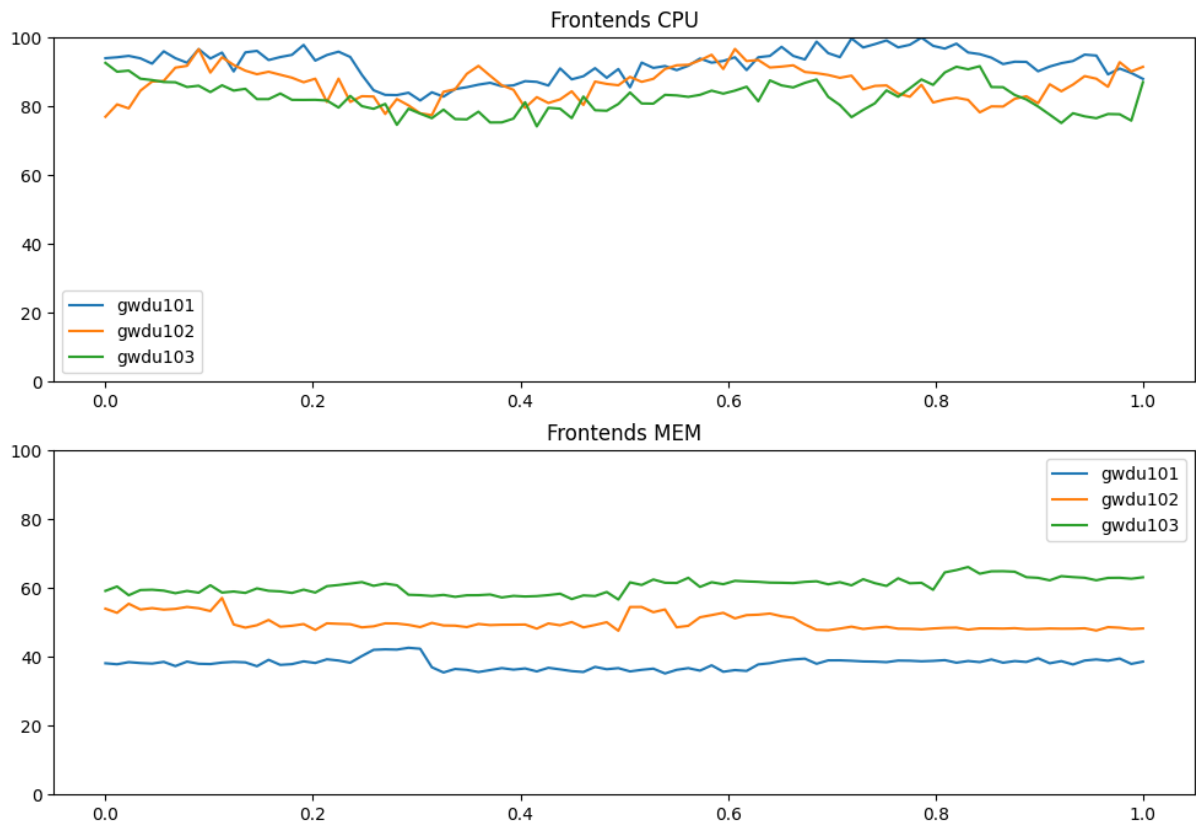


Figure 51: Test Case 1: Mocked Frontend Data

Storage Server

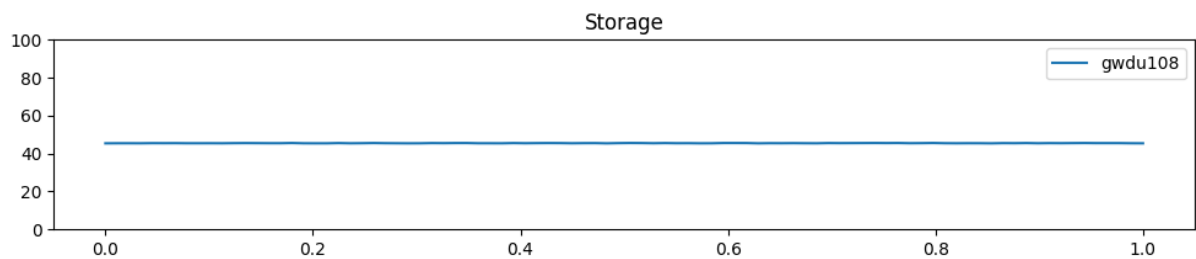


Figure 52: Test Case 1: Mocked Storage Data

CPU Nodes

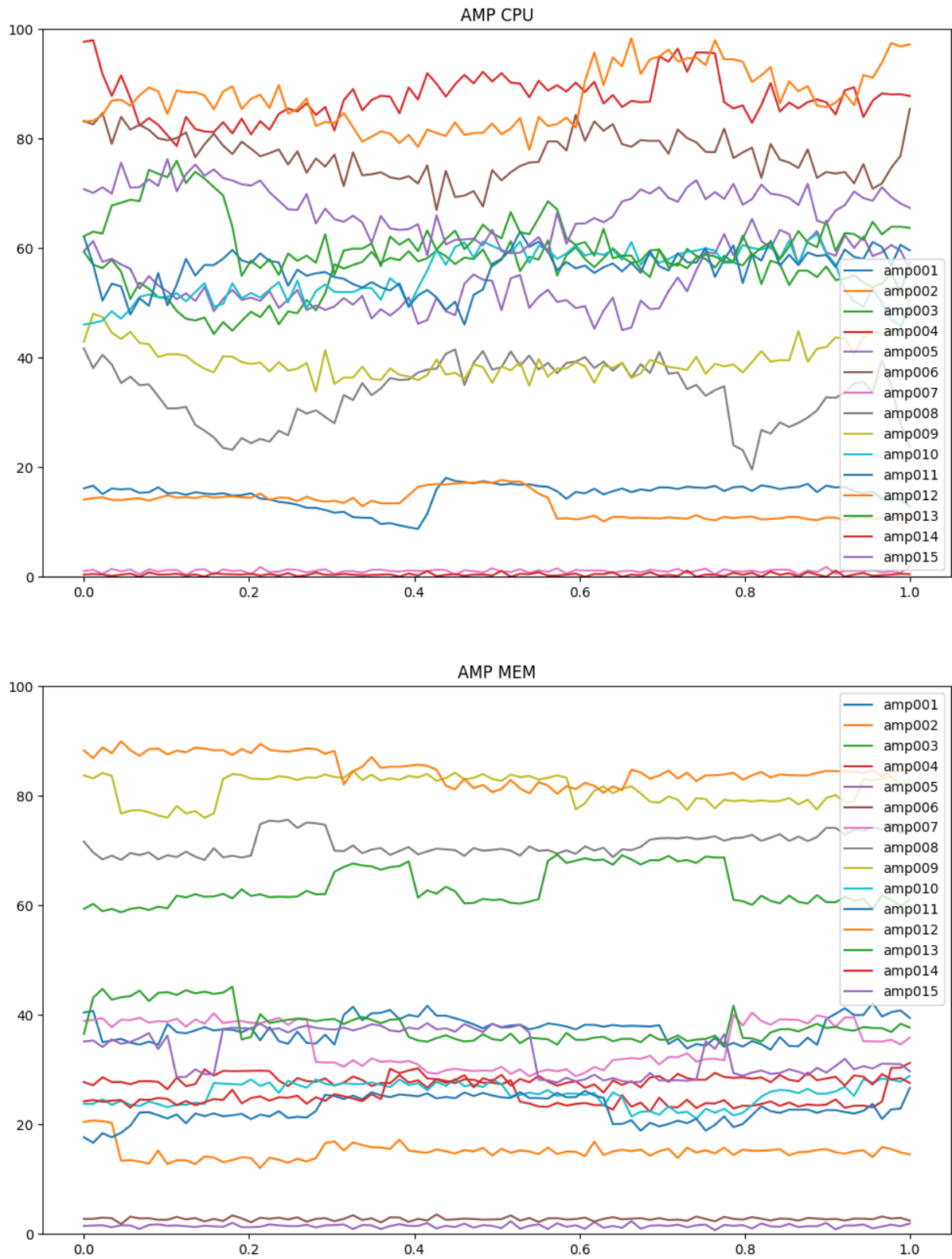


Figure 53: Test Case 1: Mocked CPU Node Data

GPU Nodes

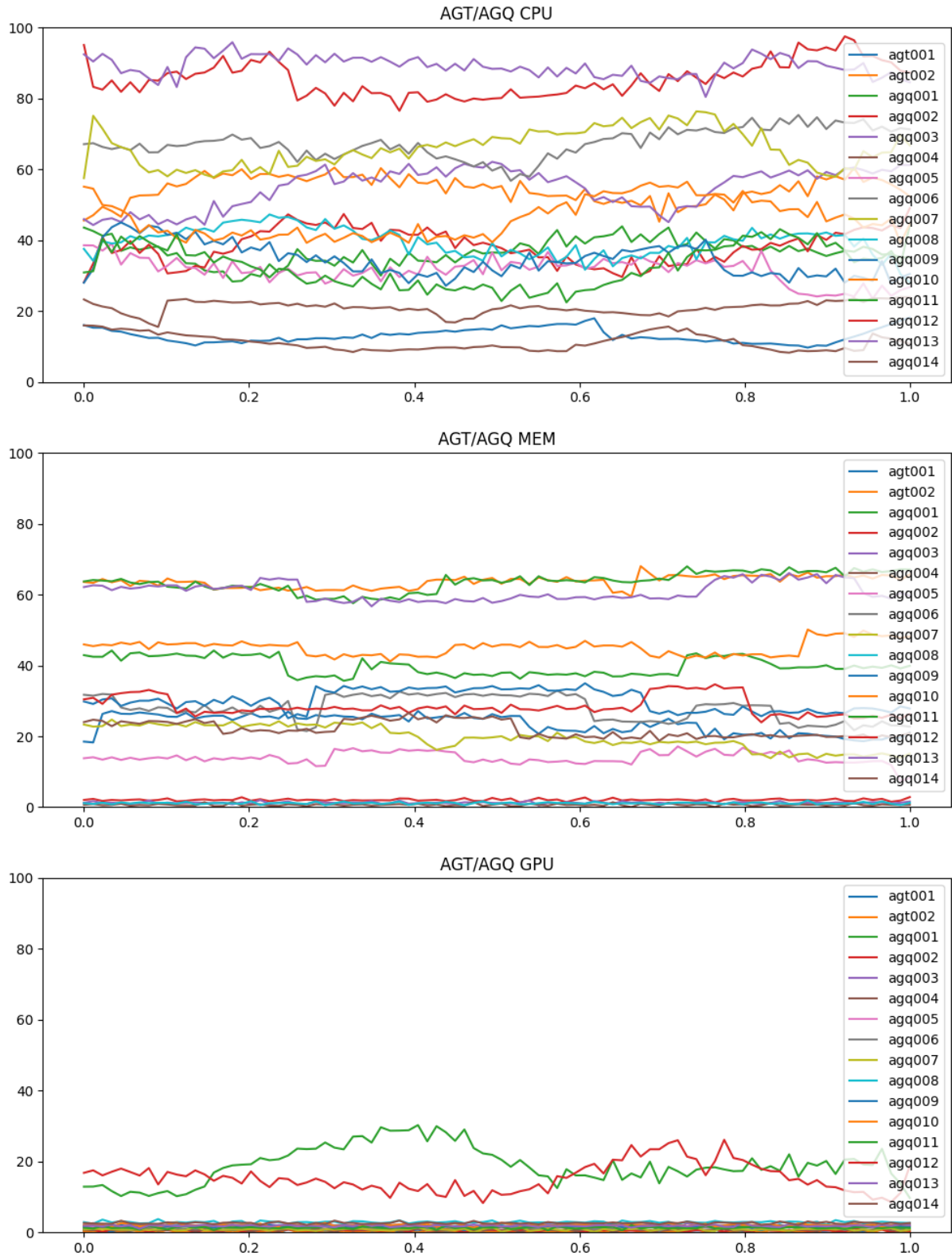


Figure 54: Test Case 1: Mocked GPU Node Data

C.2 Test Case 2: How does the scheduler work?

Frontend Servers

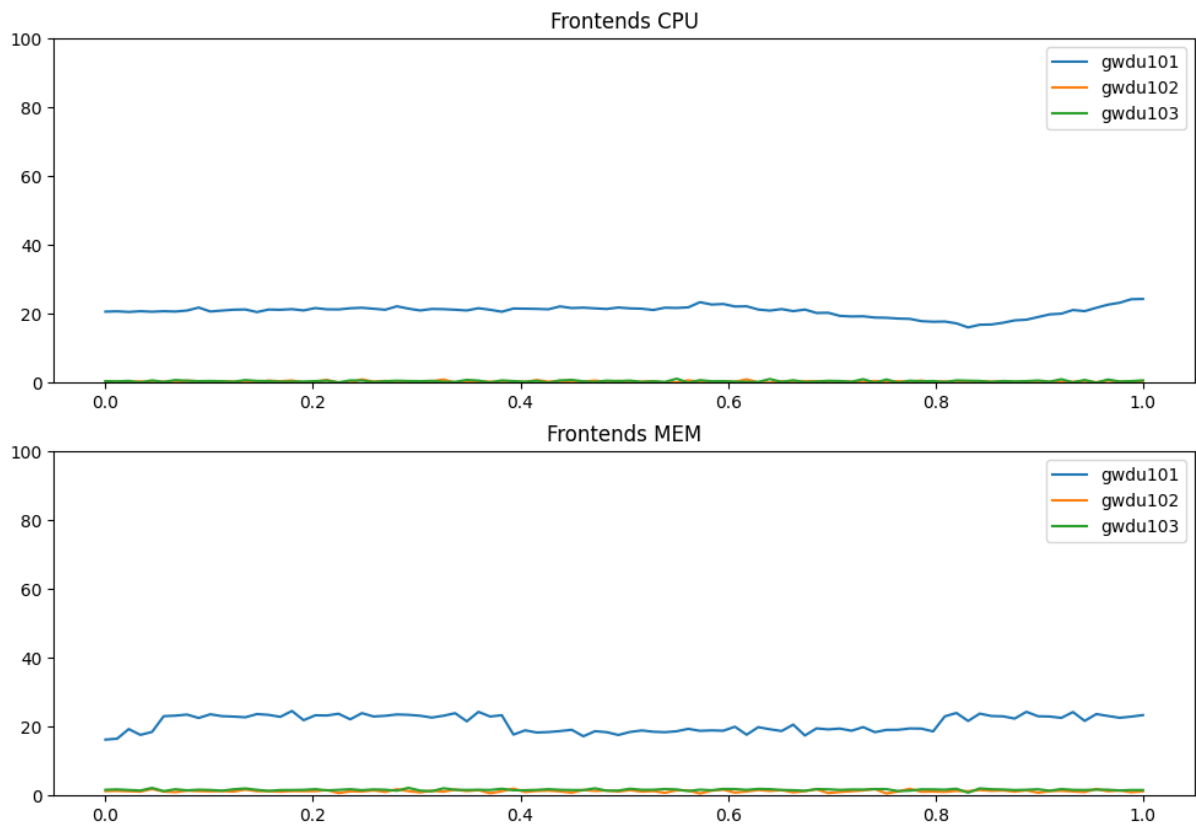


Figure 55: Test Case 2: Mocked Frontend Data

Storage Server

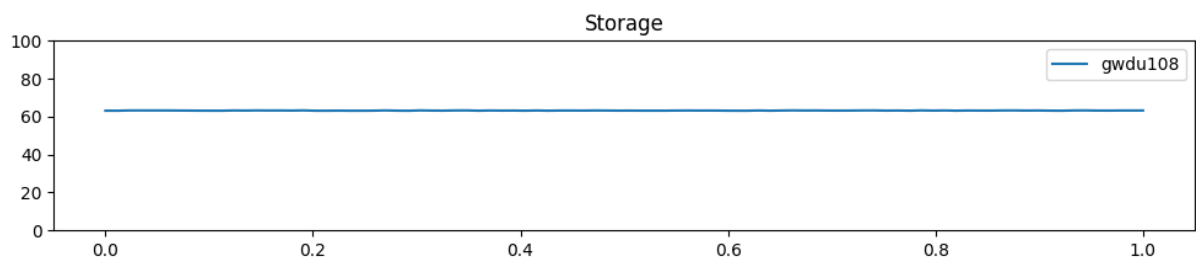


Figure 56: Test Case 2: Mocked Storage Data

CPU Nodes

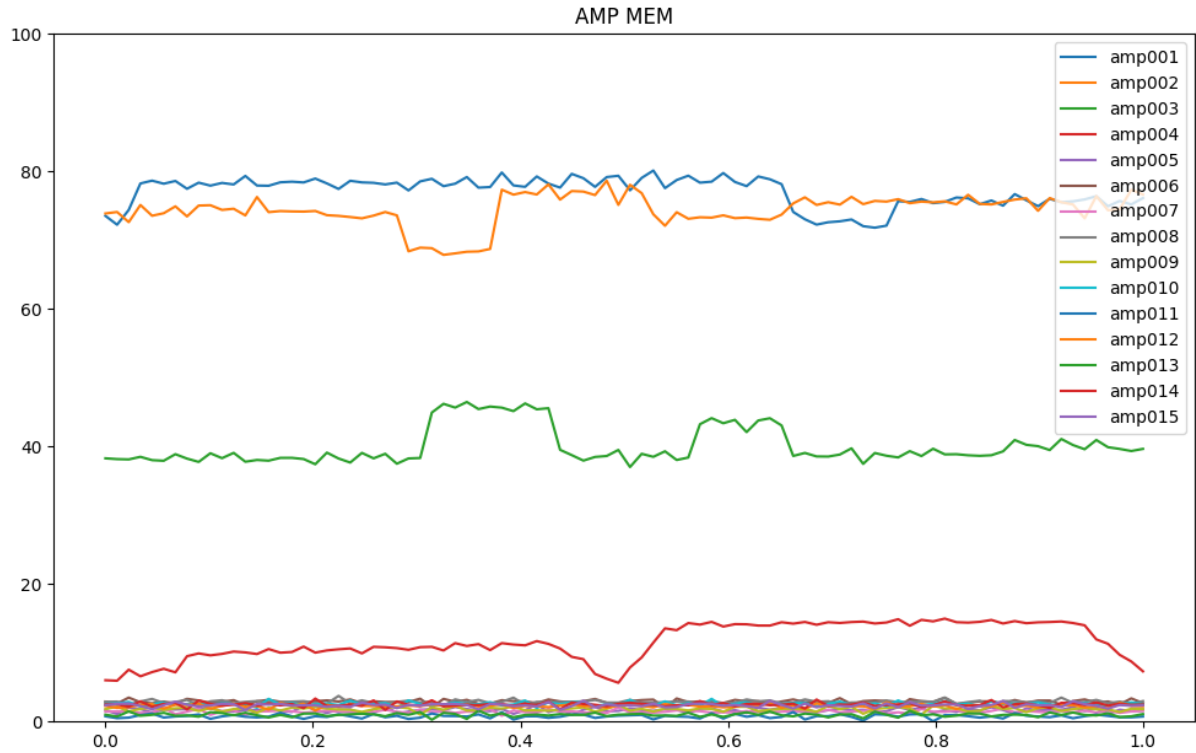
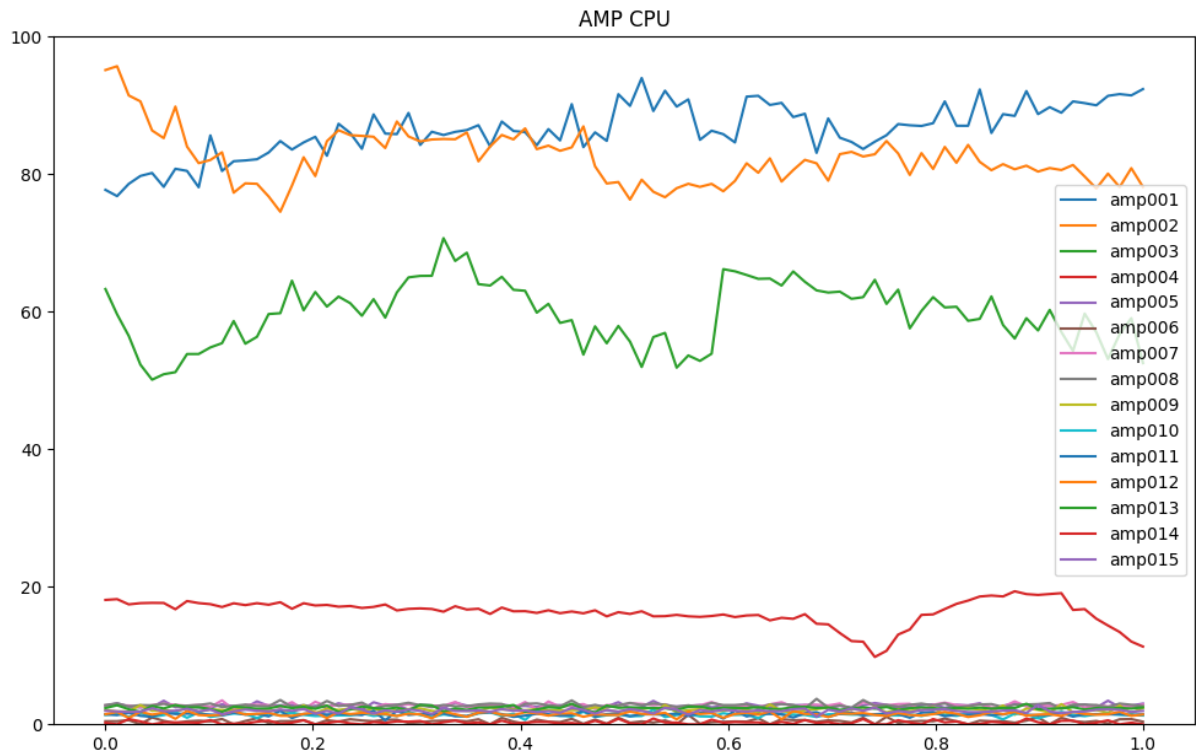


Figure 57: Test Case 2: Mocked CPU Node Data

C.3 Test Case 3: Long Term Decision Making (Data Set 1)

Frontend Servers

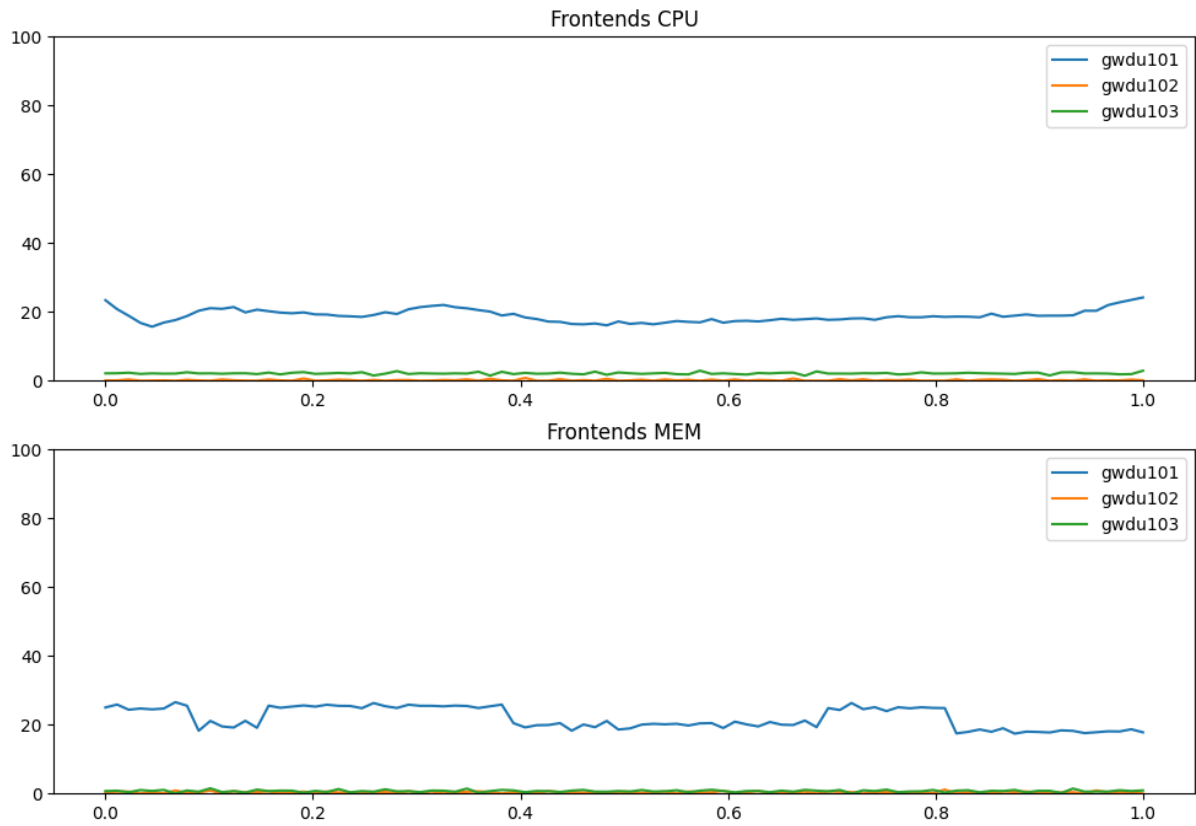


Figure 58: Test Case 3.1: Mocked Frontend Data

Storage Server

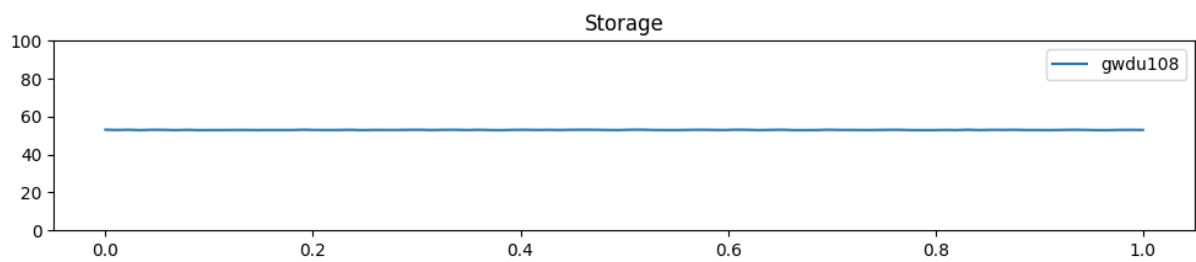


Figure 59: Test Case 3.1: Mocked Storage Data

CPU Nodes

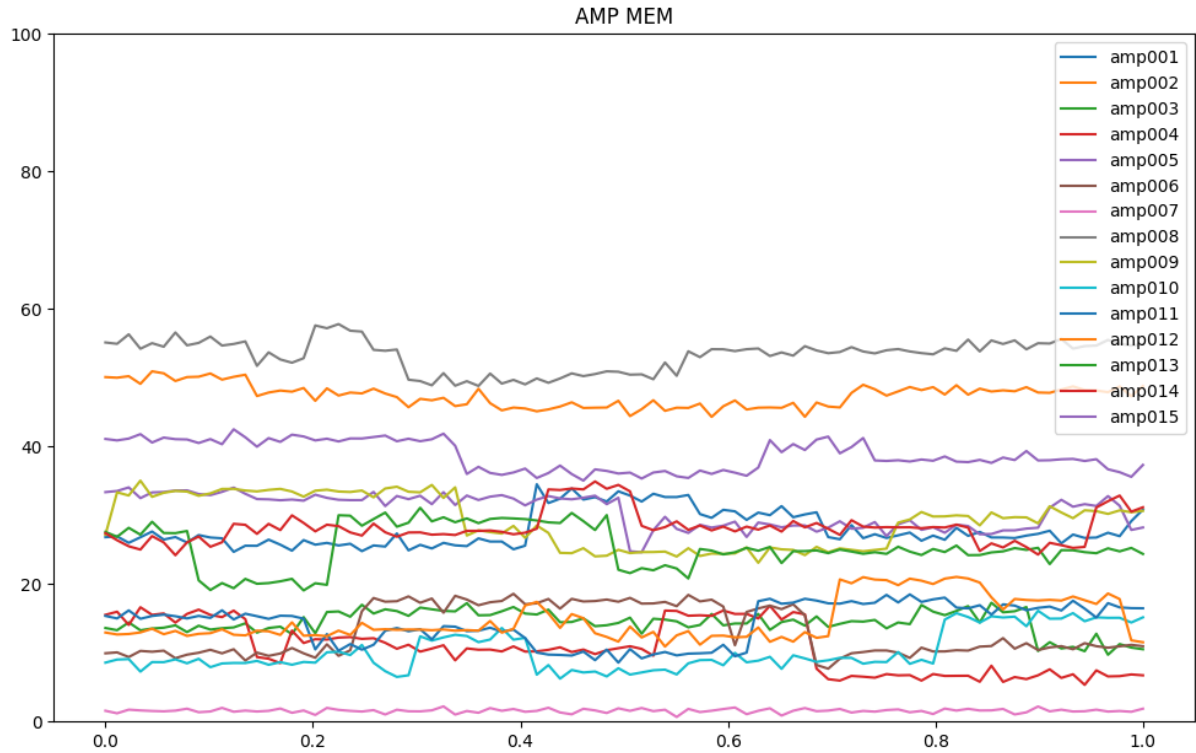
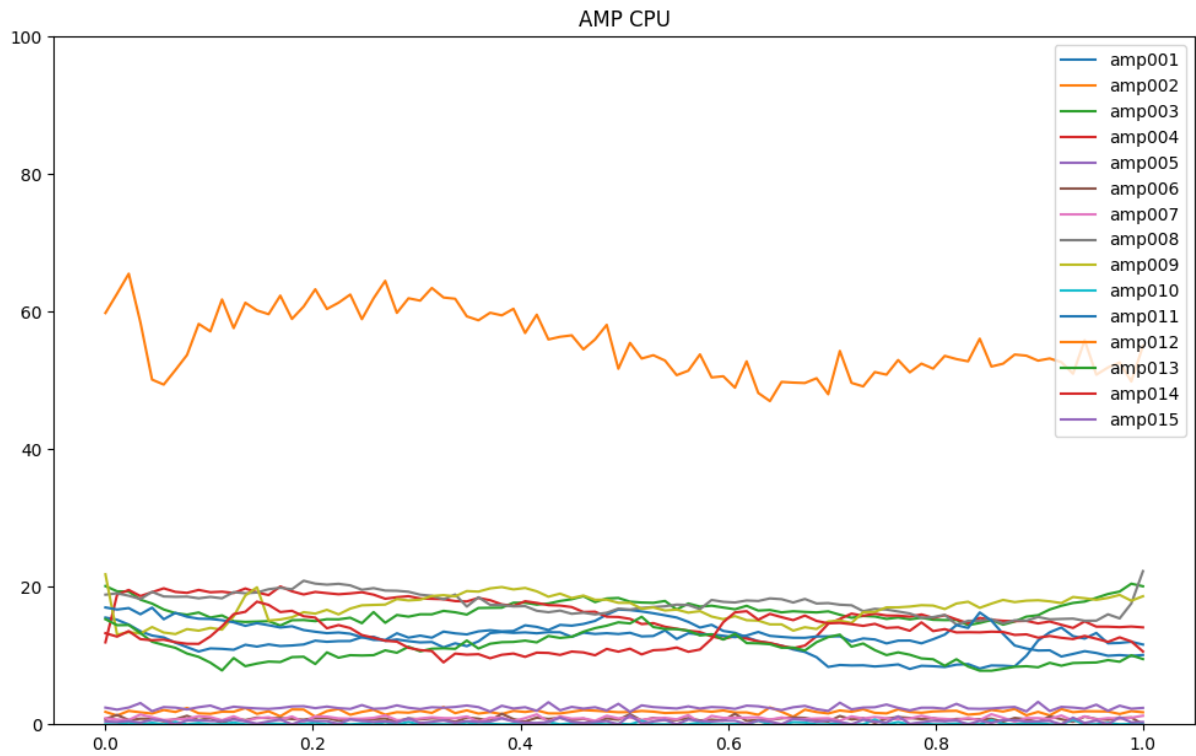


Figure 60: Test Case 3.1: Mocked CPU Node Data

GPU Nodes

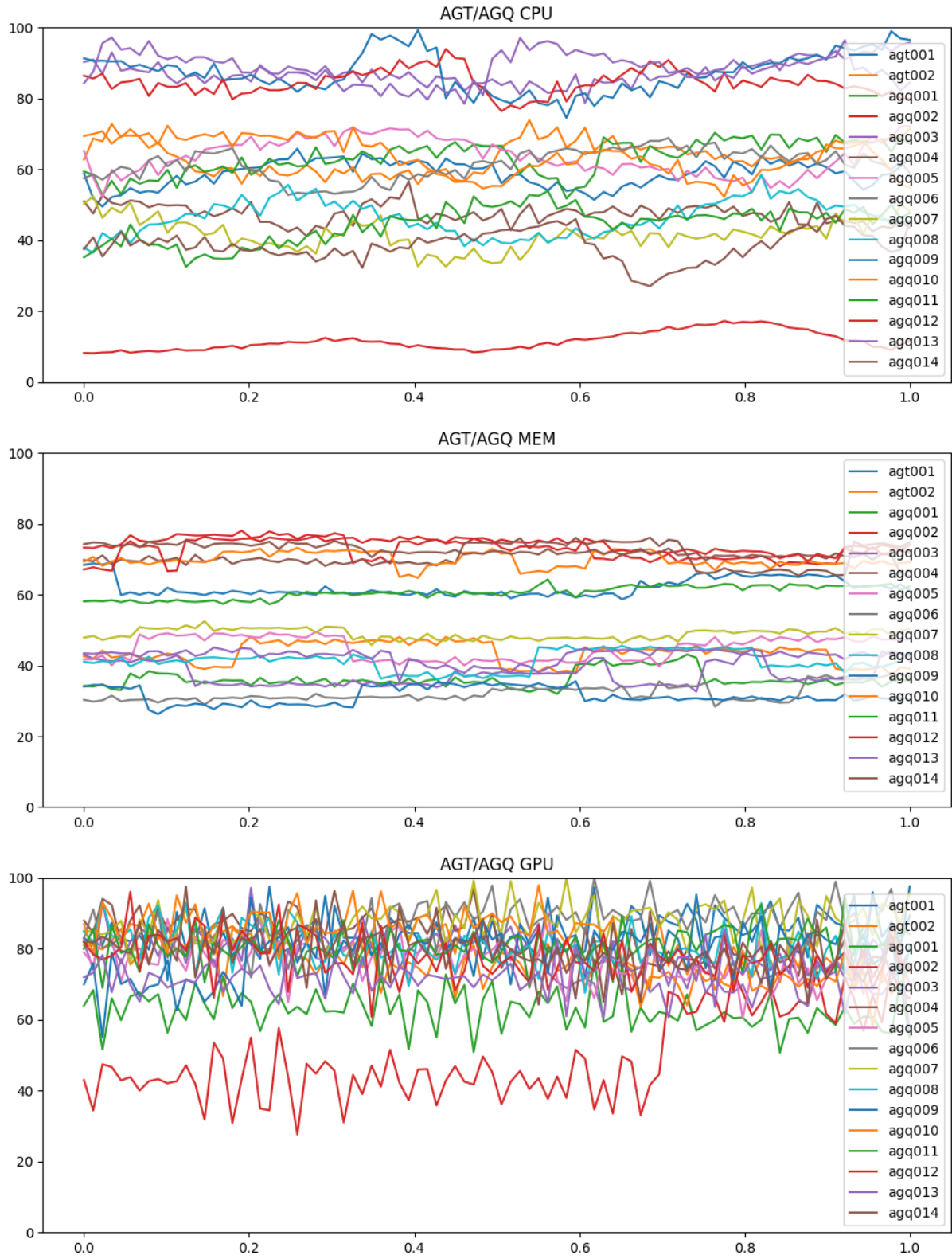


Figure 61: Test Case 3.1: Mocked GPU Node Data

C.4 Test Case 3: Long Term Decision Making (Data Set 2)

Frontend Servers

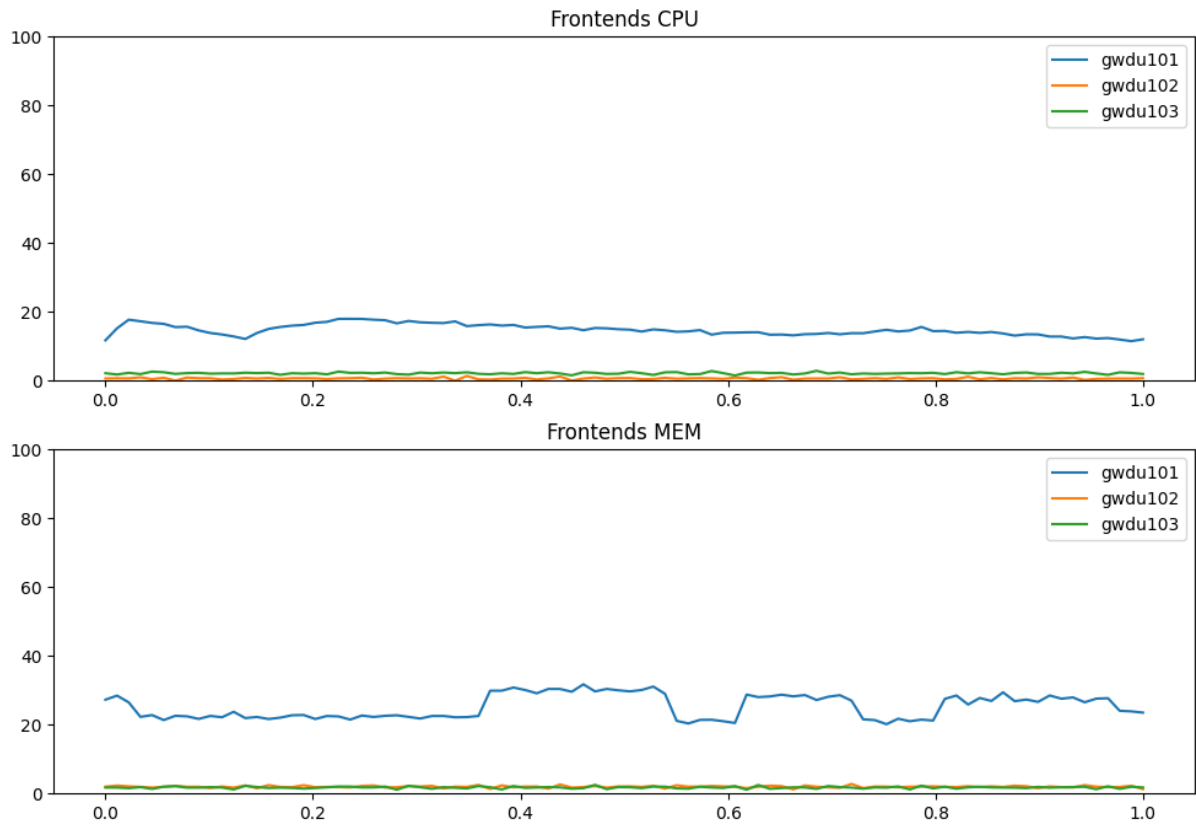


Figure 62: Test Case 3.2: Mocked Frontend Data

Storage Server

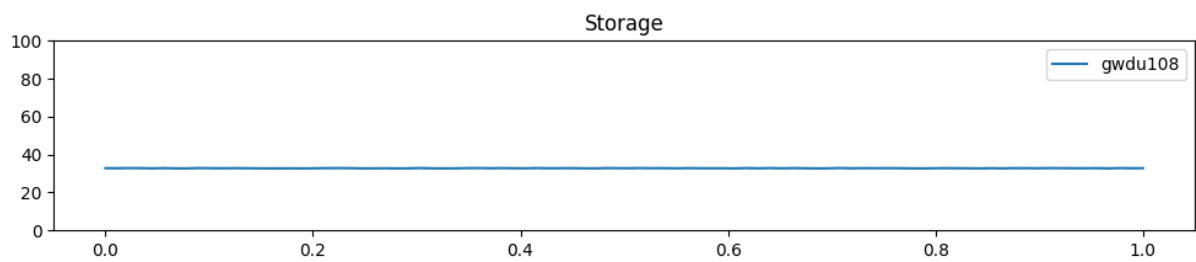


Figure 63: Test Case 3.2: Mocked Storage Data

CPU Nodes

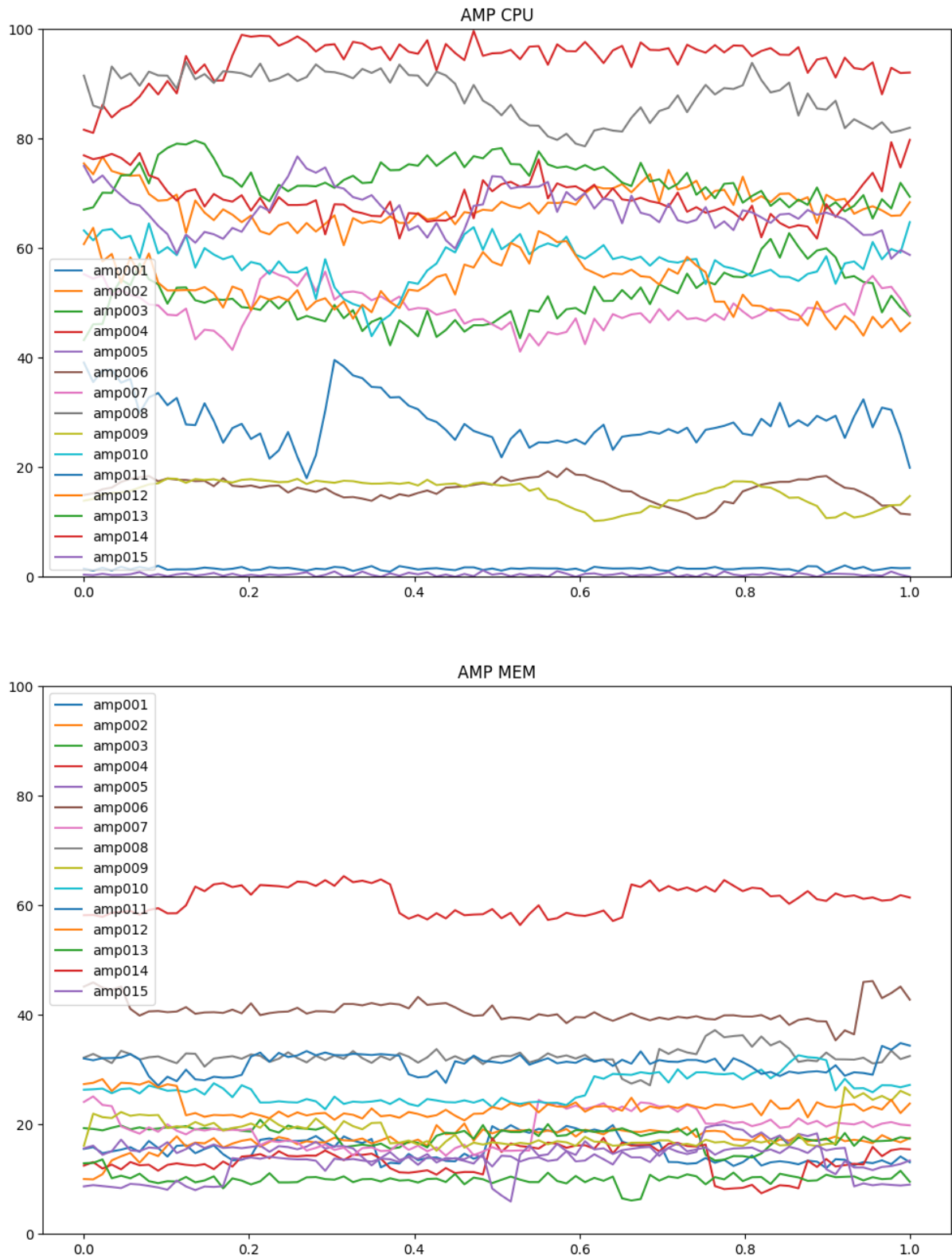


Figure 64: Test Case 3.2: Mocked CPU Node Data

GPU Nodes

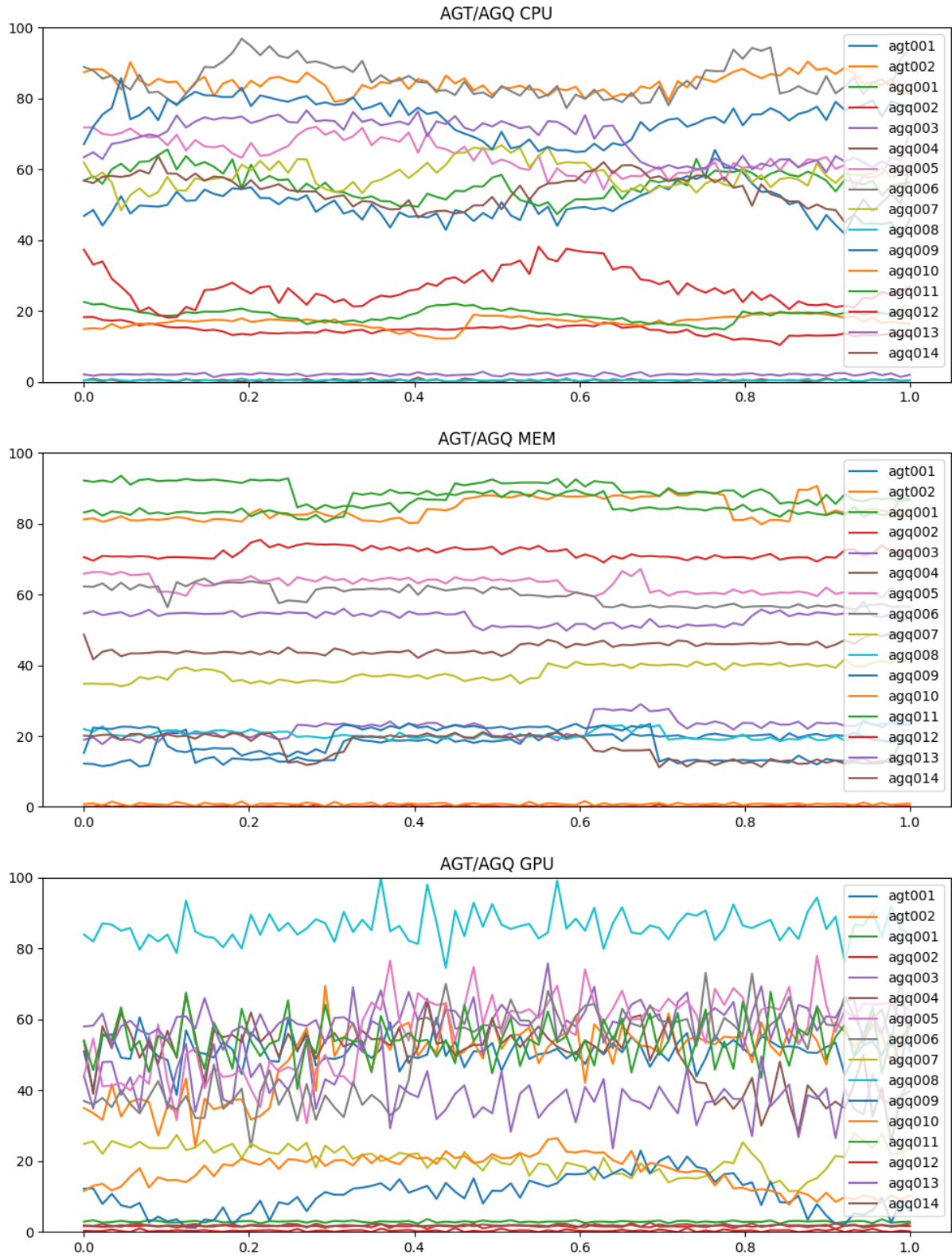


Figure 65: Test Case 3.2: Mocked GPU Node Data