Vrije Universiteit Amsterdam

Bachelor Thesis

Meterstick: a Benchmarking Tool for Performance Variability in Cloud Deployed Minecraft-like Games

Author: Jerrit Eickhoff (2641983)

1st/daily supervisor: ir. Jesse Donkervliet
2nd reader: prof. dr. ir. Alexandru Iosup

A thesis submitted in fulfillment of the requirements for the VU Bachelor of Science degree in Computer Science

June 24, 2021

Contents

1 Introduction **3**
- 1.1 Problem Statement . . . 4
- 1.2 Research Questions . . . 5
- 1.3 Main Contributions . . . 6

2 Background **6**
- 2.1 Modifiable Virtual Environments (MVEs) . . . 6
- 2.2 Cloud Computing . . . 7
- 2.3 Resource Allocation and Scheduling . . . 7
- 2.4 Yardstick . . . 8

3 Design of Meterstick **8**
- 3.1 System Requirements . . . 8
- 3.2 Design Overview . . . 9
- 3.3 Configuration Parameters . . . 10
- 3.4 Metrics Collected . . . 11

4 Implementation **12**
- 4.1 Deployment Instrument . . . 12
- 4.2 Control Servers and Clients . . . 13
- 4.3 Metric Collection Instruments . . . 14
- 4.4 Yardstick Player Emulation . . . 15

5 Experimental Setup **15**
- 5.1 Systems Tested . . . 15
- 5.2 Experiment Workloads . . . 16
- 5.3 Experiments . . . 18

6 Evaluation **19**
- 6.1 Overview . . . 19
- 6.2 Cloud environments introduce performance variability . . . 20
- 6.3 Player behaviour strongly influences performance variation . . . 22
- 6.4 Limited CPU resources affect Minecraft-like games differently . . . 24
- 6.5 Cloud environments increase variation in round trip time . . . 25
- 6.6 Complex worlds marginally increase performance requirements . . . 26

7 Discussion **27**
- 7.1 Related Work . . . 27
- 7.2 Limitations . . . 28

8 Conclusion **28**

9 Artifacts for Reproduction **29**

A Hardware Used in Cloud Deployed Minecraft-like Games **33**

Abstract

Minecraft is one of the best selling game of all time, and has inspired the development of a multitude of games with similar features, known as Minecraft-like games. Prior research has shown that Minecraft-like games do not scale well to large amounts of concurrent players. Many server operators choose to deploy using cloud computing services, either to gain access to powerful hardware or to have a stable and persistent network configuration. Additionally, prior research has theorized that the scalability of Minecraft-like games can be improved by using serverless operations to compute some game operations on the cloud. Despite the benefit of access to large amounts of resources, cloud computing has been shown to result in high amounts of performance variability. Measuring and reducing performance variability is crucial to real time services such as Minecraft-like games, as performance instability can severely harm player experience. However, there is currently no available tool to measure the performance variability of cloud deployed Minecraft-like games, making it difficult to effectively compare the performance of different cloud computing providers as well as measure the benefit of serverless scaling techniques. In this work, we design the Meterstick benchmark and use it to measure the performance variability of commercially available cloud computing services, and how this variability effects Minecraft-like games. We find that cloud computing environments introduce variability to the operation of Minecraft-like games both over time and between iterations, and that the extent of this variability is dependent on the choice of Minecraft-like game and cloud computing provider.

Keywords cloud computing, benchmark, Minecraft, distributed systems, online gaming.

1 Introduction

The revenue of the global gaming industry grew throughout 2020 to around 180 billion USD, a value greater than the global movie and TV revenue combined [38]. As a large sector in the entertainment industry, video gaming has become an integral part of every day life, as well as a communicative tool with vast international influence. However, video games as a online service represent a largely unsolved optimization challenge. The player experience of many popular online games is dependent on real-time networking with very small response times, and there is no set protocol that can be used for all game networking.

A game of particular interest in both the industry and field of network optimization is Minecraft. As one of the best selling game of all time [33] and reporting over 131 million players per month [32], Minecraft is a game with undeniable societal influence. This is especially true as Minecraft has not only functioned as an entertainment medium, but as a platform for social movements with a broad spectrum of causes, from seeking to build the entirety of Earth [4], visualization tools for metabolic pathways [17], to creating safe spaces for autistic players [2, 27], and facilitating an in-game haven for press freedom to fight against real world censorship [25].

Additionally, Minecraft's popularity has inspired a variety of new games with similar features, known as *Minecraft-like games*. Notable examples are No Man's Sky [12], Astroneer [29], Terraria [24], and Space Engineers [15].

However, the social potential of Minecraft-like games is limited by lack of scalability.

The multiplayer services of Minecraft-like games operate using a server-client architecture, in which a majority of game operations are calculated within the server and then sent to clients. Each server acts in isolation with no interactions with other servers. Prior research has shown that Minecraft-like game servers do not scale well, managing to support at most a few hundred players, and usually showing performance decreases before that maximum [35]. This is opposed to other massively multiplayer games, where concurrent players on one server can reach many thousands, and the server instances can be seamlessly moved between [39].

Server operators have the choice to host using commercially available cloud computing providers. Mojang itself, the company that develops Minecraft, chose to host Minecraft Realms, their server hosting service, on Amazon Web Services [36][1] rather than their own hardware. While they have not stated the reasoning for doing so, it can be assumed this decision not taken for horizontal scalability as Realms has a per-server max player limit of 10. Instead, this decision enabled quick vertical scaling when deploying servers for new users.

From a consumer, rather than corporate, perspective, the option of cloud deployment instead of self-hosting would be taken to gain access to more computational power without a large budget, or to have a stable network configuration that allows players to connect without going through a host's home network. While there is no available data on the amount of server operators choosing to host on cloud environments, the popularity of doing so can be seen in the growing plethora of cloud computing services specifically marketed for Minecraft or other Minecraft-like games. An incomplete sampling of these services is available in Appendix A.

Aside from the existing practice of hosting the entire Minecraft-like game server in a cloud environment, research on new scalability techniques for Minecraft-like games has theorized that computing specific sections of the Minecraft-like service in the cloud using serverless computing providers can increase performance [7].

Thus, many individuals and corporations host Minecraft-like games in commercial cloud environments, and this amount may grow further if serverless techniques prove viable and cost effective. However, while cloud computing does provide benefits in the forms of computational power and network stability, it also introduces a set of new challenges. Commercial cloud computing services operate largely as a black box, with the user of the service having no way of telling how many other users the service is being shared between, or how resources are allocated between them. This lack of transparency is crucial, as it has been shown that cloud computing can lead to high amounts of performance variability [34, 5, 16, 30].

1.1 Problem Statement

Since Minecraft-like games are real-time services, they require short and consistent response times to maintain quality of service for players. This means that measuring performance variability is important to the operation of a Minecraft-like service. However, despite the number of Minecraft-like services that utilize cloud computing, there is no currently available tool to benchmark the performance variability of cloud deployed

[1] In early 2021 Mojang migrated Realms entirely to Microsoft Azure.

Minecraft-like servers. Thus there is no easy method of comparing the performance benefits of different cloud computing services.

1.2 Research Questions

Accordingly, in this thesis we aim to answer the following main research question: *what is the performance variation introduced to Minecraft-like games when operating in cloud environments?* We then split this main research question into three research questions:

RQ1 (Section 3): How to design a benchmarking tool for cloud-deployed Minecraft-like games
To allow fair and reproducible comparison between the performance variation profiles of both Minecraft-like games and cloud computing providers the design of a benchmarking tool is required to be applicable to many Minecraft-like games and cloud providers. As the implementation and interfaces of these systems vary, the design of such a system is not a simple process.

RQ2 (Section 4): How to realize this benchmarking tool
There are no standardized methods of translating a systems design in a prototype, but the implementation needs to address important challenge applicable to benchmarking in the context of distributed systems, relating to synchronization, networking and reproducibility. This includes implementation details that allow for easy extension of the benchmarking tool for new Minecraft-like games or cloud providers.

RQ3 (Section 6): What is the performance variation of Minecraft-like games operated in real-world cloud environments
The exploration of this question requires first finding a representative set of experimental variables. This includes choice of Minecraft-like systems, cloud deployment environments, hardware within those cloud providers, and the workloads that are used within the Minecraft-like games, such as choice of world and player behaviour. Finding what instances are representative of a variable in a real world setting is challenging, as in many cases there is no quantitative source to provide an objective answer. Once these experimental variables have been decided upon, it is then necessary to design an experimental method that measures the impact of these variables using the benchmarking tool.

When exploring these research questions, we utilize the following research methods:

M1 For **RQ1** we apply the iterative AtLarge design methodology to create a benchmarking tool design [13], as well as methodologies specific to benchmarking that guide choice of metrics, workloads and baselines [23, 11].

M2 As there is no standard method of converting a systems design into a prototype, we use the AtLarge prototyping methodology for **RQ2** to iteratively implement the Benchmarking tool piece by piece, continuously integrating segments of the tool to solve the realization challenge.

M3 We answer **RQ3** by using an experimental method that evaluates appropriate quantitative system and application level metrics in controlled, reproducible, but still representative settings [11].

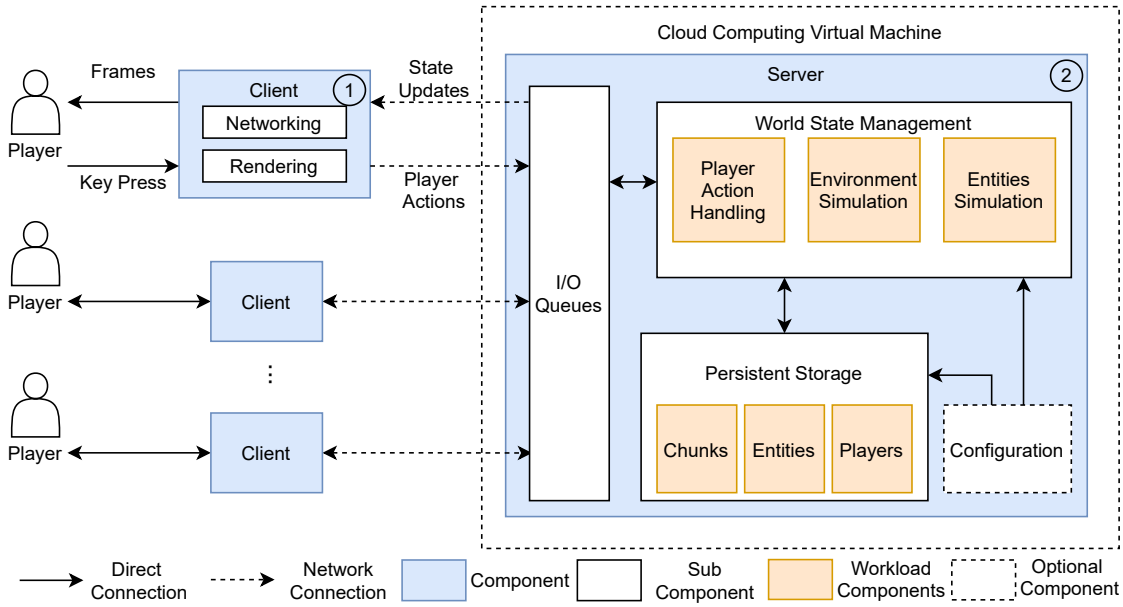


Figure 1: Overview of a Minecraft-like MVE system.

1.3 Main Contributions

Following from our set of research questions and methodologies, the contributions of this thesis are as follows:

C1 Define performance variability in the context of cloud-deployed Minecraft-like games.

C2 Design a tool to measure performance variability of cloud deployed Minecraft-like games.

C3 Implement a prototype of this tool and show its efficacy.

C4 Evaluate the performance variability profile of different cloud computing services and Minecraft-like services.

2 Background

In this section we introduce relevant background information relating to Minecraft-like games and the operation of cloud environments.

2.1 Modifiable Virtual Environments (MVEs)

Minecraft-like games include any video game that utilizes both a server-client multiplayer system and a Modifiable Virtual Environment.

A *Modifiable Virtual Environment* or MVE is a type of game world that allows the player to freely add or remove sections. Often these systems are voxel-based, comprised of discrete blocks that can be created or removed. Additionally, MVEs generally incorporate

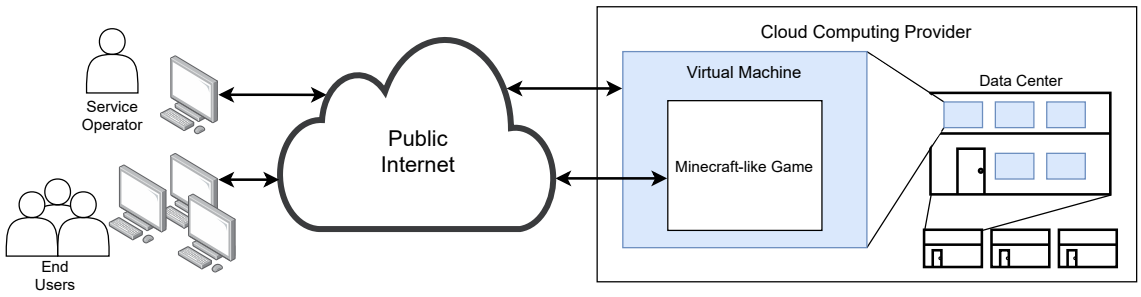


Figure 2: Cloud computing in the context of Minecraft-like games.

some form of dynamic nature, with sections of the environment changing over time without direct player influence. This is used to facilitate common game features such as simulating gravity, lighting or pathfinding.

Figure 1 depicts the general structure of a MVE service. A majority of the computation for an MVE in a Minecraft-like game is done by the server (component 2), rather than the client (component 1). The creation, modification, and maintaining state consistency of an environment is all done server-side. This centralized structure is common, even ubiquitous, in the implementation of MVE services due to the modifiable property of the environment. It is not possible to make assumptions on the state of the environment at any given time, at any given moment large sections of the world can be changed. Thus, other architectures, such as peer-to-peer, may introduce prohibitively high amounts of overhead and latency from having to duplicate world data upon each update and maintain state consistency between users.

2.2 Cloud Computing

Cloud computing is the practice of having sections or the entirety of programs be run on virtual machines within distributed computer systems, usually a data center with a large amount of computational resources. We show a simplified overview of this architecture in Figure 2. This setup allows users without access to necessary computational power or network stability to run intensive workloads. There are many cloud computing providers in operation, with the largest by market share being Amazon Web Services, Microsoft Azure, Google Cloud and Alibaba Cloud [26]. A key reason to use these Infrastructure as a Service (IaaS) providers is the ease of resource management: virtual machines can be created or extended quickly with minimal interruption to existing programs.

2.3 Resource Allocation and Scheduling

This elasticity in resource management that IaaS promises can cause problems. Because these cloud computing services have many users with varying computational needs that may change rapidly, all commercial IaaS providers utilize automated scheduling policies to allocate computational resources [5, 16, 30]. However, the information on what allocation

policy is used and how many users the resources are being shared between is not provided to the end users. This means it is difficult for any user to make any assumptions on available computational power at any given time, the cloud computing provider could be overloaded and throttling some processes to allow preferred ones to execute, or could be under capacity and able to give a given process all necessary resources.

Adding to the difficulty, cloud computing performance is heavily reliant on the internal networking of the data center, and even computationally trivial processes can generate large amounts of network traffic between computing clusters [34]. Additionally, it is challenging to predict when a service will require more resources, as often this is dependent on external societal factors, such as the scheduling of an social event or even failures within an unrelated datacenter causing services to balance load to other locations.

2.4 Yardstick

The Yardstick project [35] created a benchmark tool to evaluate the scalability of Minecraft-like games by connecting increasingly large amounts of emulated players and observing how network and system metrics changed.

Within the Yardstick benchmark there is a component to facilitate the emulation of players with differing in-game behaviour, we extend and adapt this component for use in Meterstick (see Section 4.4).

3 Design of Meterstick

In this section we introduce the design of Meterstick, our benchmarking system for performance variation in cloud deployed Minecraft-like games. We define a set of requirements, and discuss both the system design and design considerations.

3.1 System Requirements

We outline 8 design requirements. The first three are requirements specific to the benchmarking of cloud deployed Minecraft-like games. The last five relate to the benchmarking of computer systems in general and are taken from existing guidelines [37, 14].

R1 *Reproducible measurement of performance variability:* Though the benchmarking tool measures performance variation over time, the results must still be reproducible such that the same system with the same workload gives similar results.

R2 *Validity of workloads:* The workload of the server, including the network load, in-game world, and player behaviour must be accurate to real world settings.

R3 *Completeness of experiments and metrics:* There are many variables involved in the operation of a cloud deployed Minecraft-like game. The benchmark should be able to operate different experiments in order to isolate and measure many of these variables, with suitable metrics.

R4 *Fairness:* The benchmark should provide a fair assessment for compatible systems. In particular, bias towards any one system should be limited.

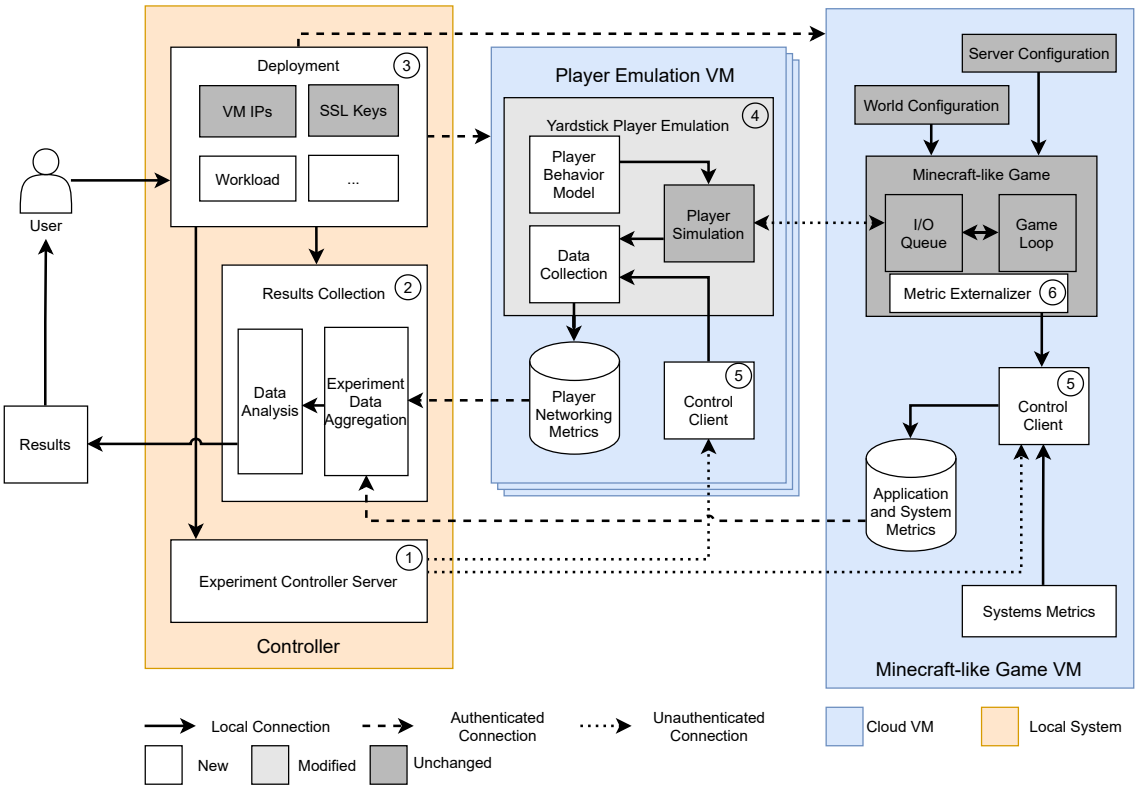


Figure 3: Design Overview of Meterstick

R5 *Ease of Use:* The benchmark should be easy to configure and use such that applying the benchmark to new compatible systems is simple for future researchers.

R6 *Clarity:* The benchmark should choose and present results in a way that is logical to display the performance of the target system.

R7 *Representativeness:* The benchmark should be comprised of workloads and metrics representative of real-world use cases.

R8 *Portability:* All segments of the benchmark should run on different cloud platforms.

3.2 Design Overview

We give an architectural overview of Meterstick in Figure 3. As we design a benchmarking tool, we additionally define a benchmarking technique, which includes the experiment configuration (Section 3.3), metrics to measure, and how to measure them (Section 3.4).

The steps of the benchmarking process and the corresponding design of Meterstick fulfill the requirements listed in Section 3.1. Meterstick is capable of testing any Minecraft-like game that is compatible with the Minecraft protocol and metric externalizing tool (see Section 4.3), and as such is both fair (**R4**) and easily extensible to new systems (**R5**).

In order to operate in a cloud environment, the design utilizes a deployment tool (component 3) that facilitates the operation of both the Player Emulation and Minecraft-like game via the use of the established networking protocols. File copying and systems

Parameter	Step	Effect
IPs	Deployment	Changes what nodes Meterstick connects to
SSL Keys	Deployment	Authenticates Meterstick
Servers	Deployment	Changes what Minecraft-like games are tested
World	Deployment	Changes what world the servers run
File Locations	Deployment	Changes where deployment sends files
Resume	Deployment	Specifies to resume a previous experiment
Ports	Metric collection	Specifies what IP ports Meterstick uses
JMX URLs	Metric collection	Specifies JMX URLs for metric externalization
JMX Ports	Metric collection	Specifies a range of JMX ports
RAM	Experiment	Specifies the max RAM passed to JVM
Number of Bots	Experiment	Specifies how many players will be connected
Player Behaviour	Experiment	Changes emulated player behaviour
Duration	Experiment	Changes how long an iteration lasts
Iterations	Experiment	Specifies how many iterations will be run

Table 1: List of configurable parameters. Step refers to the stage of the benchmark process that is effected.

level concerns are operated through an authenticated connection, and the operation of nodes during the experiment is controlled by a small unauthenticated server (component 1). Thus, the tool is capable of benchmarking on any set of nodes that support suitable networking protocols, as well as the programming languages Bash, Python and Java. As this is true on any modern cloud provider, Meterstick is portable (**R8**).

To measure the performance of Minecraft-like games under a realistic workload (**R7**, **R2**) Meterstick connects emulated players that move realistically (component 4), and runs the Minecraft-like game with a representative world workload (discussed in Section 5.2).

To ensure that the benchmarking results are both logically displayed and reproducible (**R6**, **R1**) Meterstick is capable of supporting many types of experiments to isolate specific variables (**R3**). Since Meterstick's design utilizes a controller server and client system (components 1 and 5), the behaviour of each node can be remotely controlled and easily adapted for each experiment, and experiments can be extended to many nodes.

3.3 Configuration Parameters

Meterstick is highly configurable, a list of configuration parameters are given in Table 1. By adding servers into the servers folder, new systems can easily be added. Similarly, since the deployment process simply copies the servers folder to the remote node, the world that the Minecraft-like games run can easily be configured by copying in a new world folder.

The experiment can be configured by specifying the duration, number of iterations, what servers to run, how many players to connect and the behaviour of those players. Since the IPs of the nodes that the deployment tool connects to can be anywhere that is network accessible, nodes can be chosen that are geographically distant or within the same datacenter in order to measure (or avoid) the performance impact caused by public network infrastructure.

3.4 Metrics Collected

The common metric for judging the performance of a Minecraft-like game is tick time. A *tick* is the measurement of time for one iteration of the game loop. Within this loop, all Minecraft-like games compute a series of environment processes, receive packets from clients, and then send state change packets to all clients. For each Minecraft-like game, there is a threshold in tick time where the game loop takes too long and the server cannot compute all state updates it needs to send to the clients. This results in stuttering, rubber-banding, and lag for players. For as long as tick times are above this threshold, the Minecraft-like game is considered in an *overloaded* state. Because tick time is an internal measurement of Minecraft-like games, it is necessary to externalize it so it is visible to metric collection instruments.

Another application level metric measures server performance by observing the round trip time between server and client. Minecraft-like games are sensitive to network delay. The client section of a Minecraft-like game performs few game operations itself, instead sending the result of player input to the server and rendering the state updates that the server responds with. This means that the in order for a given player action to be processed and then results displayed for the player, the action must be sent over network to the server to an input queue, processed by the server, its result put in an output queue, and then sent back over the network to the client. Thus, every action has a delay of more than twice the network latency between the client and server. We measure this round trip time by observing the elapsed time between a sent chat message and the servers response (see Section 4.4)

Additionally, systems level metrics such as CPU utilization, network activity, process thread counts, and disk I/O are collected on the node running the Minecraft-like game to avoid relating performance solely to application level metrics.

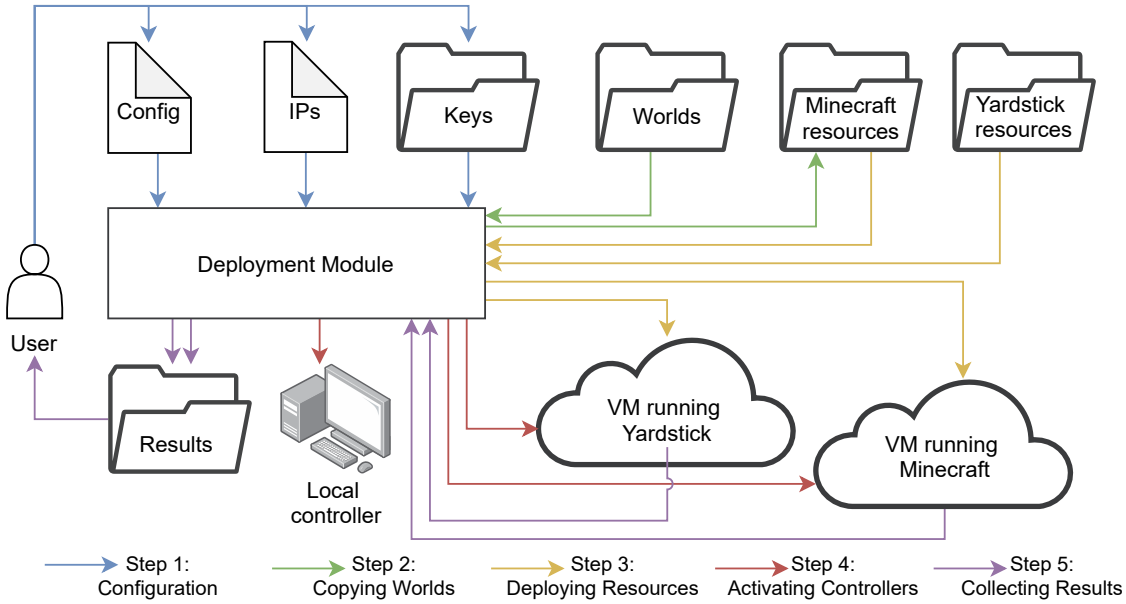


Figure 4: Deployment module operation by step.

4 Implementation

In this section we discuss the implementation details of the Meterstick design shown in Section 3 (with a focus on components 1, 3, 5, and 7). We implement Meterstick as three main modules: deployment, control, and metric collection. We discuss each in turn, and then describe our changes to the Yardstick tool.

4.1 Deployment Instrument

The deployment script is written in bash, we depict its operation in Figure 4. The deployment process has five main steps, in the first configuration parameters, IP addresses and SSL keys provided by the user are read from file. In the next step the relevant worlds are copied to server folders. Then in step three the Minecraft and Yardstick resources are copied to their respective remote VMs via SCP, and in step four the control servers and clients are activated through SSH. Finally, in the final step, which takes place after the experiment has concluded, the deployment module copies the results to the local file system for analysis by the user.

As cloud environments are complex, errors from configuration, deployment, networking and authentication are common. Thus, the deployment tool has the ability to resume a previously canceled experiment, provided the user-supplied configuration and IP addresses have not changed. It does so by first retrieving partial results, then calculating how far the previous run had gotten, and resuming from the cutoff point.

Message	Effect	Dest	Ack
set_server:*server*	Specifies name of server	YS/MC	Yes
set_jmx:*jmx url*	Specifies JMX URL	MC	Yes
iter:*iteration*	Specifies what iteration to start at	YS/MC	Yes
initialize	Starts the selected server	MC	Yes
log_start	Starts metric logging tools	MC	Yes
log_stop	Stops metric logging tools	MC	Yes
stop_server	Stops running server	MC	Yes
connect	Starts player emulation	YS	Yes
convert	Converts metric bin files to CSV	YS	Yes
ok	Acknowledges the previous message	Controller	No
keep_alive	No-op, keeps TCP connection open	MC/YS	Yes
err:*error*	Previous message has caused error	Controller	No
exit	Stops the controller client	MC/YS	No

Table 2: List of controller messages. Dest specifies what nodes the message is for where YS is player emulation and MC is the server node. Ack specifies if the message requires an acknowledgement.

4.2 Control Servers and Clients

The controller server is a Python script responsible for the operation and synchronization of the benchmark, as well as error reporting if one of the controller clients encounters a problem. We depict the overview of the control server and client modules in Figure 5, and the set of control messages exchanged through TCP in Table 2. The player emulation controller clients are responsible for configuring, activating, and retrieving the metrics from their respective tools.

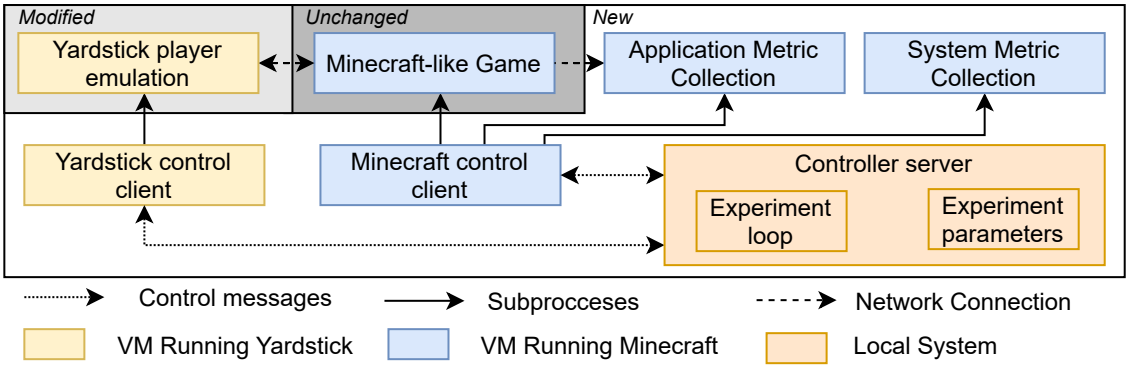


Figure 5: Control server and client module interaction.

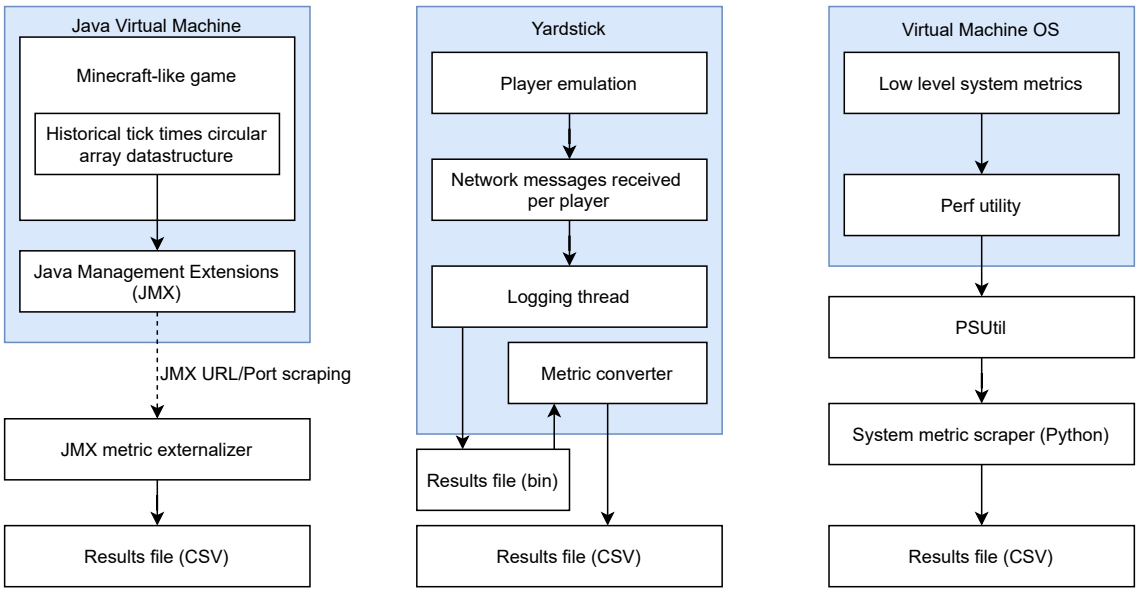


Figure 6: Metric-collection instruments operation.

4.3 Metric Collection Instruments

In Figure 6 we depict the instruments and processes for collecting systems and application level metrics.

To measure tick time (see Section 3.4), which is an internal measurement to the Minecraft-like service, it must be externalized. We do this using Java Management Extensions (*JMX*), which exposes metrics within the Java Virtual Machine by using a network API with an associated JMX URL and port. Activating JMX requires the Minecraft-like service to be started with a set of JVM arguments [22], and each Minecraft-like service has a specific JMX URL[2].

Connecting to and collecting the metrics from the Minecraft-like game server is done using a jar packaged with the java standard library JMXConnector class. Since tick time is internally stored as a circular linked-list of length 100 in the Minecraft-like service (5 seconds worth of ticks), this jar samples at a rate of once per 2.5 seconds. This way, the changes in the tick times array are the ticks that have occurred since the last sample, and the amount of new ticks can easily be discerned.

The measurement of both systems level metrics such as CPU utilization and network traffic is done using a python script with the PSUtil library [28], at a sampling rate of twice per second.

Both the JMX and PSUtil metric collection instruments continuously sample their metrics as well as associated timestamp and write these to a log file, and continue to do so until they are sent a termination signal by the controller client.

[2]For most Minecraft-like services using the Minecraft protocol, this URL is *net.minecraft.server:type=Server*, but it can easily be discovered with JConsole or other Java management tools.

4.4 Yardstick Player Emulation

We extend the Yardstick player emulation tool in two ways, we add new player behaviour to suit our experimental method (see Section 5.2) and we adapt the Yardstick network message logging to additionally capture the contents of chat messages.

To measure the round trip time between player and server at an application level we have one player send random chat messages to the server at a fixed 1 second interval frequency in both of the player behaviours we use. Yardstick logs packets that each player sends or receives to file, along with timestamp. We add to this logging functionality the ability to capture relevant chat text for each chat message that is sent or received.

Thus, we find the round trip time as the difference in timestamp between the outgoing chat message packet and the server response with the same message.

5 Experimental Setup

In this section we outline the design of experiments to measure performance variation using Meterstick.

5.1 Systems Tested

We run experiments on three environments and measure the performance variability of three Minecraft-like games.

5.1.1 Minecraft-Like Games

We select three Minecraft-like games that use the Minecraft protocol: Minecraft itself, *Forge*, and *PaperMC*. Our selection criteria used to decide upon these services is both popularity and utility.

We chose the official, "*Vanilla*," Minecraft server [21], as it is the default service that most users and hosts operating Minecraft servers employ. This includes users of the server hosting service *Realms*, which is advertised inside the Minecraft client [20]. Notably, the Vanilla service does not allow for gameplay modifications of any kind.

The second is the Forge server, which extends the Minecraft protocol with an API supporting community made gameplay modifications ("*mods*"). Forge is the most popular server for operating modified services [10]. Of the top 50 most downloaded Minecraft mods, only 5 are not exclusive to the Forge API, and of those 5, only 1 is not available on Forge [6].

The third is the PaperMC server, which we chose as it is marketed as a high-performance alternative to the Vanilla server [31]. PaperMC does not support mods, but it does support server-side *"plugins"* that modify the functioning of the game without changing the Minecraft protocol. While the PaperMC project provides no quantitative figures on what performance gain it gives compared to the Vanilla server, it does provide a list of optimizations, including extensive changes to threading models and environment processing.

5.1.2 Cloud Computing Providers

The commercial cloud providers used in the experiments are Amazon Web Services (*AWS*) and Microsoft Azure. We decided on these services based on their percentage share of the cloud computing market, where AWS has 32% and Azure has 20% [26]. Additionally, to provide a controlled environment baseline to compare against, experiments were run on the DAS-5, a distributed super-computer for academic and educational use [3].

On AWS and Azure it is possible to select specific configurations of hardware. To ensure that the configurations we use in the experiments were representative of real world settings, we selected options based on both official and community recommendations. For a full list of sources that motivated our choice of hardware, see Appendix A.

On AWS we use a "T3.Large" node, which offers two vCPUs and 8GB RAM [1]. On Azure we use a "Standard_D2_v3" node, which also provides two vCPUs and 8GB RAM [18]. Nodes were selected within the same subnet to reduce variation caused by network delay (except for the experiments where this was measured).

The DAS-5 nodes used each have 16 cores and 64 GB RAM (though RAM usage is limited to 4GB by the -Xmx JVM argument in all experiments).

5.2 Experiment Workloads

In Minecraft-like games, the *workload* of a server is a combination of players and world. In this section, we outline what workloads were used in the experiments and our selection criteria for representative workloads.

5.2.1 Emulated players

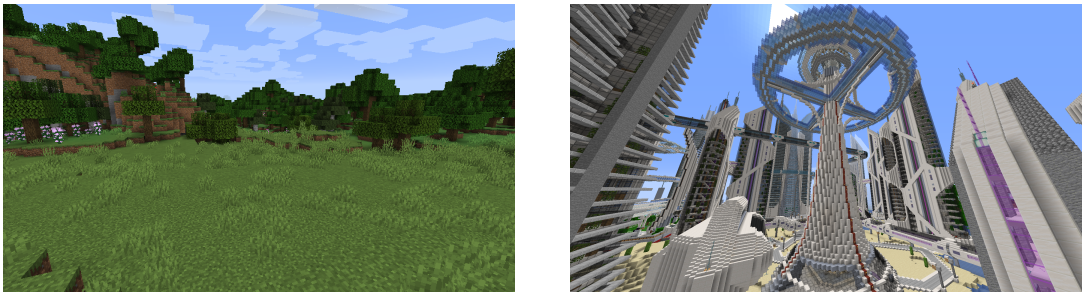
The player contribution to the workload of the server comes from both amount of players and the behaviour those players exhibit.

For player count, we use a fixed number of 25 players in each experiment. We do not have access to data on the average number of players on dedicated server instances, and the data that is available on server player count generally reports the total number across combined networks with many servers, rather than single servers. Instead, this number is selected from the Minecraft Wiki's dedicated server recommendation [19] as well as the recommendations from the Minecraft server providers listed in Appendix A.

We run experiments with two types of player behaviour: bounded random and run away. These two behaviour types were picked to be both representative and reproducible. With bounded random, each player moves continuously within an area of fixed size. In each experiment using this behaviour, this bounding area size was set at 32 voxels wide. Additionally, the seed of the random generator is fixed, such that on the same map every iteration yields the same player movements.

With the run away behaviour type, each player is assigned a direction vector out of 8 compass directions. For the duration of the experiment, each player moves quickly in that direction. In order to avoid getting stuck on obstacles, the direction vector is randomly adjusted by a small margin when the player gets stuck. This behaviour also uses a fixed seed for random generation, so multiple iterations yield the same behaviour.

Additionally, in both the bounded random and run away behaviours, a single player is selected to stand still at the location they are added into the game world. We use this



(a) Baseline world *(b) FutureCity world*

Figure 7: Screenshots from worlds used in experiments.

player's network message data to allow comparison between experiments with different player behaviour.

5.2.2 World

As all Minecraft-like games utilize a largely procedurally generated modifiable virtual environment, the specific layout of that environment must be controlled when benchmarking. In our experiments, we run the Minecraft-like services with two different game worlds: a baseline, freshly generated world with no modifications, and a community made map named "Future City." Figure 7 shows in-game screenshots of these worlds.

The performance requirements of the MVE change by what world they operate. To have a control world with minimal performance impact that is still a realistic use case, we create a 'baseline' world. This world is freshly generated by the Minecraft-like game with the default options (and seed: -392114485) and has not been modified.

To analyze the impact that the world has upon Minecraft-like game performance variability, we also run experiments with a more complex map: a community made world called "Future City." This map has 492 thousand downloads on the website MinecraftMaps and has a size of 113MB [40]. For comparison, the baseline world is 5.4 MB.

The Future City map contains many complex structures as well as many player-made "Redstone" devices, that add dynamic functions to the world. For instance, there is a large, functional clock in the center of the city built in the map. Such dynamic systems should increase the resource requirements of this world.

5.3 Experiments

Name	Section	Variables Changed	Variation source measured
Internal Variation	6.2, 6.4	None	Over time
External Variation	6.2	Iterations, Duration	Between repetitions
World	6.6	World	World workload
Behaviour	6.3	Behaviour	Player behaviour
Network	6.5	Location	Network delay

Table 3: List of experiments. Section refers to where we analyse experiment results.

Parameter	Values
Hardware	**DAS-5, AWS, Azure**
MVE	**Vanilla, Forge, PaperMC**
World	**Baseline**, FutureCity
Behaviour	**Bounded Random**, Run Away
Players	**25**
Iterations	**1**, 50
Duration	**300 seconds**, 50 seconds
RAM	**4GB**
Locations	**Paris**, California

Table 4: Experimental parameter configuration. Values in bold are used unless otherwise indicated.

We give a list of experiments in Table 3. We run experiments by first establishing a baseline experiment that measures the Internal Variation over time. Subsequent experiments then change single variables to measure isolated impact on performance variation. We outline the experimental parameters in Table 4, where experiments use the values in bold unless otherwise indicated.

6 Evaluation

In this section we present and analyze the data collected from running the experiments in Table 3.

6.1 Overview

Our main findings are:

MF1 Minecraft-like games exhibit high amounts of performance variability in real-world cloud environments (Section 6.2). We find that cloud environments introduce performance variation both over time and between iterations, up to a 100ms variance in server tick time. Of the cloud environments tested, Azure introduced the most variability.

MF2 Player behaviour significantly increases performance variation of Minecraft like games (Section 6.3). When emulated players run away in all directions, each Minecraft-like game becomes increasingly more overloaded over time, in both cloud environments and dedicated hardware. In the Azure environment, each server tested exceeds maximum tick time by up to 250ms.

MF3 Limited CPU resources effect each Minecraft-like game differently (Section 6.4). Representative cloud environments have limited vCPU count, which causes Minecraft-like games to utilize 50% to 75% fewer threads compared to dedicated hardware. We observe that the Minecraft-like games tested show two main behaviours in the presence of this reduced thread count.

MF4 Cloud environments increase variation in round trip time (RTT), especially when the Minecraft-like game is in an overloaded state (Section 6.5). In the Behaviour experiment, where each server becomes increasingly overloaded, there is higher variation in RTT between players and the server. In the Forge and Vanilla servers on AWS, the interquartile range of RTT grows to 50ms from the 20ms in the Internal Variation experiment. Additionally, we find that geographical distance between nodes does not increase RTT variation but introduces a fixed offset equal to twice the network latency. Finally, we find that server network bandwidth is limited by CPU utilization.

MF5 Complex game worlds marginally increase the performance requirement of MVEs (Section 6.6). When using a complex game world, each Minecraft-like game exhibits slightly increased variation in tick time of around a 15ms addition to tick time interquartile range, as well as a higher frequency of outlier ticks of high duration.

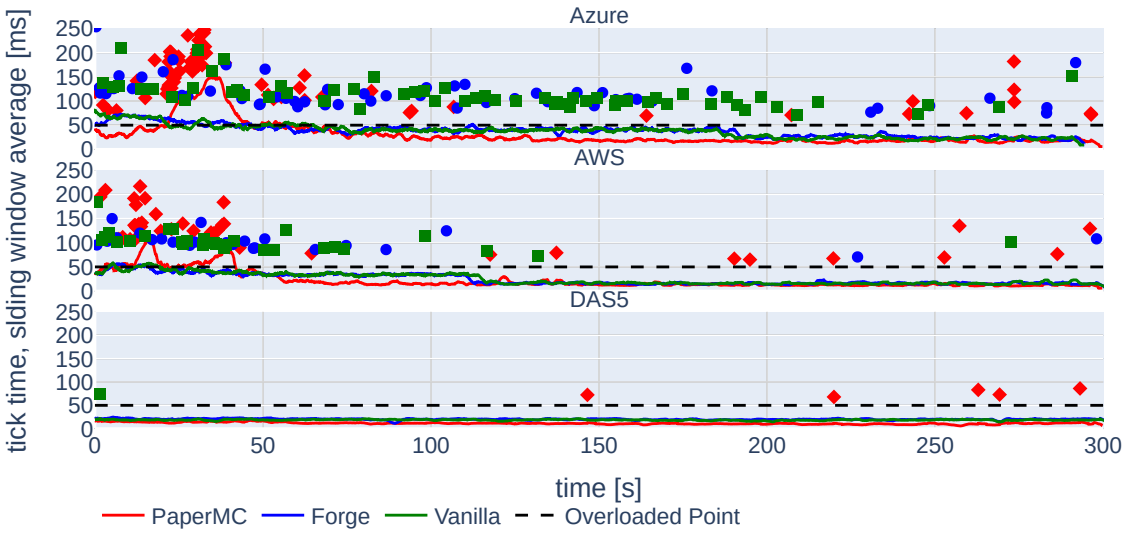


Figure 8: 2.5 second sliding window average of tick time for each Minecraft-like game and cloud provider combination over one iteration of 300 seconds. Peaks 50ms greater than the sliding average are marked.

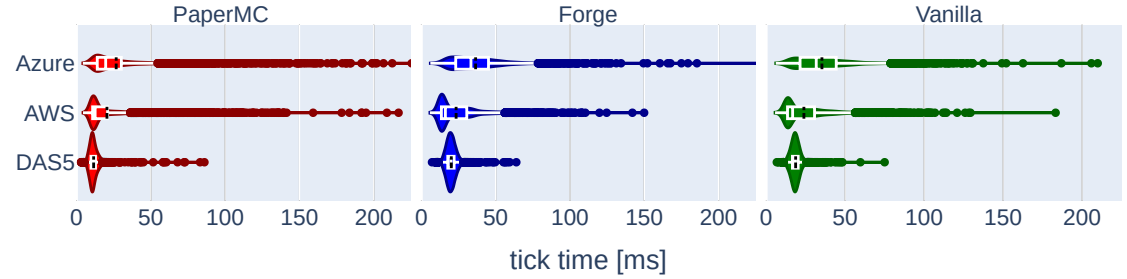


Figure 9: Violin plot of tick time over one iteration of 300 seconds. Black line depicts arithmetic mean.

6.2 MF1: Minecraft-like games exhibit high amounts of performance variability in real-world cloud environments.

In Figure 8 we depict the tick time data collected during the Internal Variation experiment. It shows that both AWS and Azure introduce performance variability over time to the operation of each Minecraft-like game, characterized by an initial period of instability. For each Minecraft-like game, a 2.5 second sliding window average of tick time is shown, as well as markers for peaks of tick time more than 50 ms above that sliding average. Also shown is the overloaded point at 50ms. When game tick durations are longer than this point the game service is said to be overloaded, and experiencing reduced quality of service.

We observe that on the DAS5 hardware, which offers dedicated compute nodes, the Minecraft-like games are largely stable throughout the experimental duration. The sliding average for each server stays less than 25ms, only infrequently peaking to higher than the

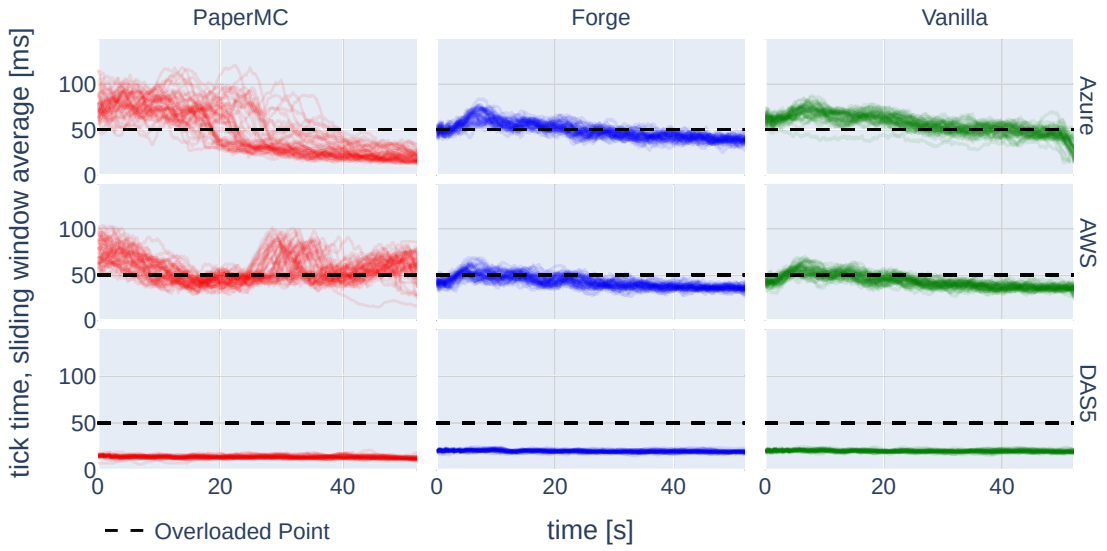


Figure 10: 2.5 second sliding window average of tick time for each Minecraft-like game and cloud provider combination from 50 iterations of 50 seconds.

overloaded point. We conjecture that this relative stability is due to the greater amount of non-shared CPU resources available on the DAS-5 nodes.

On the AWS and Azure services, all the Minecraft-like games tested had increased performance variability. In general there are two performance profiles: that of PaperMC which has an extremely unstable period of high tick times but stabilizes to sub-overloaded levels after around 50 to 55 seconds, and that of Forge and Vanilla[3], which do not have as high of an initial spike but tend not to stabilize for an extended period of time (190 seconds on Azure and 115 seconds on AWS).

Even after stabilizing to a sub-overloaded state, on both AWS and Azure there are spikes to an overloaded level at a much higher rate compared to the DAS5 control. In Figure 9 we show violin plot of the Internal Variation tick data, which depicts the extent to which AWS and Azure introduce performance variability. Where the DAS-5 only has a few points that are slightly above the overloaded point, both AWS and Azure have a high frequency of overloaded ticks at a greater magnitude.

Figure 10 shows the data from the External Variation experiment. It depicts a 2.5 second sliding window average of tick time from 50 iterations of each Minecraft-like game service and cloud provider combination. In Figure 11 we depict the violin plots of the same data. These show that both AWS and Azure introduce variability between iterations, to a different degree for each Minecraft-like game.

The DAS5 environment again shows little performance variability, each iteration stays consistently at the 20ms point, with only around 2ms of variation between iteration sliding averages. The DAS5 iterations do not show the period of instability followed by a stabilization as the iterations on cloud environments do, which we propose to be an effect

[3]Since Forge is a modified version of Vanilla whereas PaperMC is based off of a completely separate Minecraft-like game server, it was expected that Forge and Vanilla show similar behaviour, especially in the Baseline case which does not make use of features that Forge has modified in comparison to Vanilla.

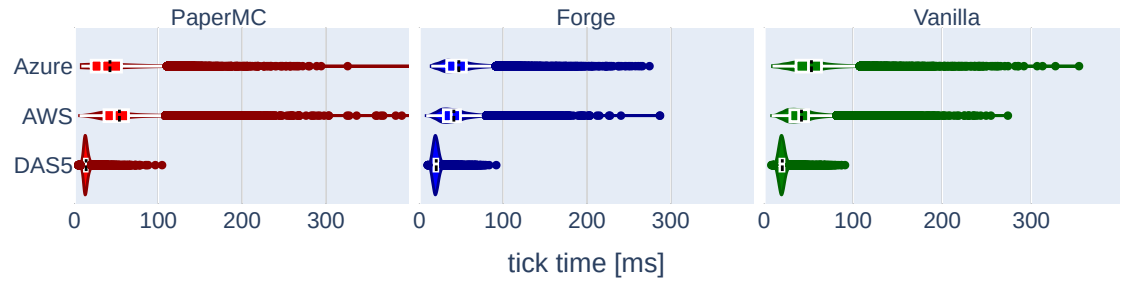


Figure 11: Violin plot of tick time over 50 iteration of 50 seconds. Black line depicts arithmetic mean.

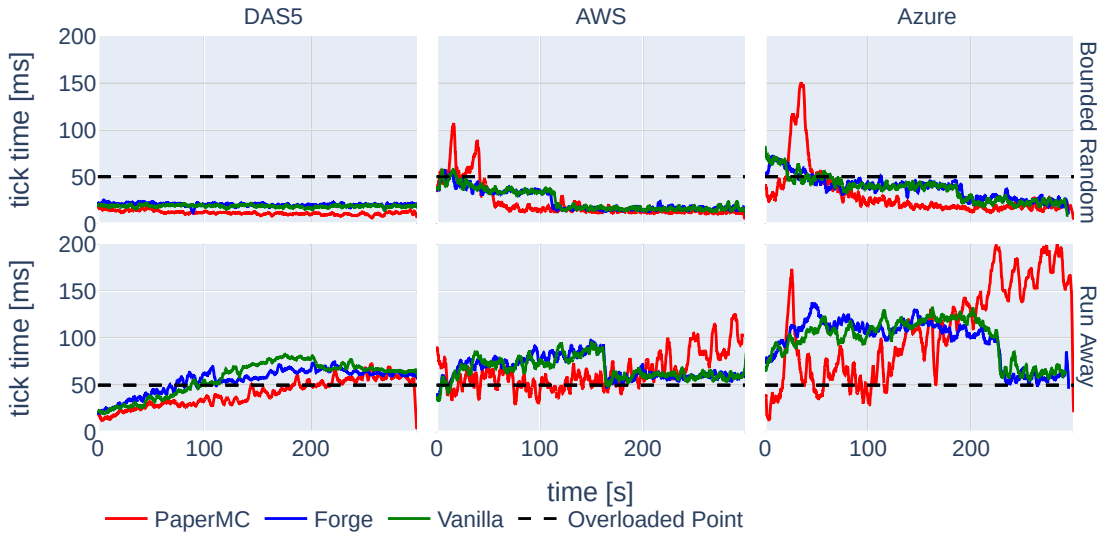


Figure 12: Effect of player behaviour on tick time over time.

of increased CPU resource available (we elaborate further in Section 6.4).

On AWS and Azure, Forge and Vanilla show an increase in variation compared to DAS5, in general the sliding averages vary by 25ms up to 50ms on outlier iterations. PaperMC shows far more variation, with up to 100ms difference between iterations on Azure and around 60ms on AWS, thus performance variation is dependent not only on cloud environment but also upon the choice of Minecraft-like game.

From both the Internal and External Variation experiments, we find that on representative real-world cloud computing environments Minecraft-like games running realistic workloads exhibit increased performance variability, and that such variability is severe enough to impact the quality of service of the Minecraft-like game.

6.3 MF2: Player behaviour significantly increases performance variation of Minecraft like games.

In Figure 12 we see the result on performance from having the emulated players using the "run away" player behaviour explained in Section 5.2. In all environments, including

the dedicated compute nodes on the DAS5, and for all Minecraft-like games, tick time increases to above the overloaded threshold and shows increased variability. After around 150 seconds, all servers in each environment are in an overloaded state.

The run away player behaviour is in some ways the worst-case scenario during the operation of a MVE server. The server must generate chunks of new terrain, serialize them, and send to each of the connected players, and since players are running in different directions the chunks they cause to be generated do not overlap. Moreover, this scenario can be reasonably expected to be common in real-world operation of Minecraft-like games, as travelling and exploration throughout the procedurally generated world is a central feature of MVEs.

In the case of PaperMC, tick time continued to increase for the entire experimental duration, whereas Forge and Vanilla had sudden drops in tick time after a few minutes. We conjecture this is due to the differences in how each Minecraft-like game creates and stores terrain, including their approach to writing new chunks to file and performing garbage collection.

The run away player behaviour caused significant increase to both the sliding window average and the variance of tick time, such that it causes overloading of each server in each cloud environment. Thus, we find that player behaviour significantly impacts the performance of Minecraft-like games.

Additionally, the fact that player behaviours such as long distance movement have non-trivial performance requirements results in a corollary finding: cloud computing environments limit possible player behaviour by causing increased performance variability.

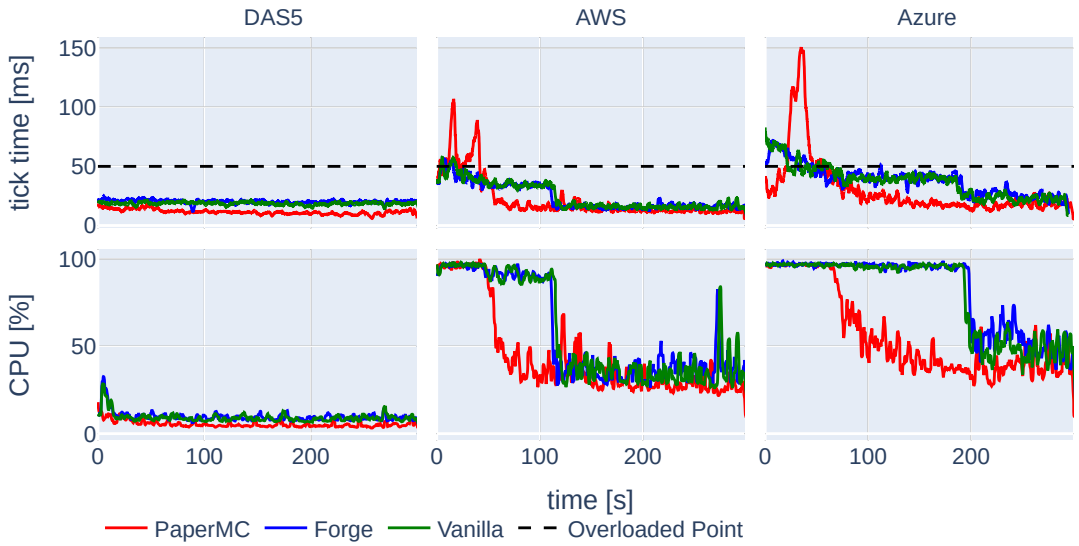


Figure 13: Comparison of tick time and CPU utilization during Internal Variation experiment.

Environment	Minecraft-like Game	Threads
DAS-5	All	105 - 115
AWS, Azure	PaperMC	59
AWS, Azure	Forge, Vanilla	40

Table 5: Number of threads used by each Minecraft-like game in the Internal Variation experiment.

6.4 MF3: Limited CPU resources effect each Minecraft-like game differently.

In Figure 13 we compare the CPU utilization and tick time for each Minecraft-like game on different cloud providers during the Internal Variation experiment. This shows that a lack of sufficient CPU resources causes different effects for PaperMC compared to Forge and Vanilla. Tick time and CPU utilization are strongly correlated on both AWS and Azure, where there is near-total utilization of the CPU resources from each server until the point at which they stabilize to a sub-overloaded tick time.

From this, we propose that periods of high tick time in this experiment may be caused by a lack of available CPU resources. This is backed by the availability of CPU resources throughout the experiment on the DAS5 environment and the correspondingly low tick times.

Table 5 gives the number of threads each Minecraft-like game uses in each environment, showing that on the DAS5 all servers utilize similar numbers of threads, but on the Azure and AWS environments Forge and Vanilla utilize 75% fewer threads compared to the DAS5, whereas PaperMC uses about 50% fewer.

For PaperMC, which uses more threads in the commercial cloud environments, we

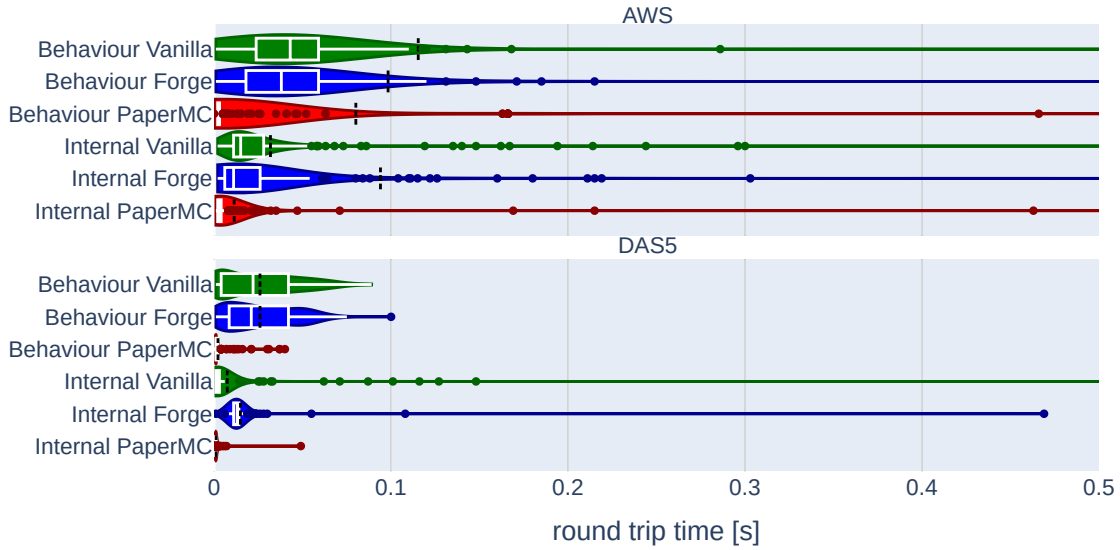


Figure 14: Round trip time violin plot for each server on DAS5 and AWS during the Internal Variation and Behaviour experiments. Black line depicts arithmetic mean.

see higher tick times for a duration of 50 seconds and then a decrease to around 25ms tick time, which corresponds to when CPU usage declines, also around 50 seconds. Vanilla and Forge, with fewer threads by comparison, have spikes in tick time before the stabilization point with less magnitude, but take more than twice as long to stabilize.

From these findings, we conjecture that due to difference between the threading schemes of Minecraft-like games [4] as well as the performance characteristics between cloud environments, the limitations to CPU resources in representative cloud environments cause different forms of variation between Minecraft-like games.

6.5 MF4: Cloud environments significantly increases variation in round trip time, especially when the Minecraft-like game is overloaded

In Figure 14 we show the statistical range of round trip time from each server on the DAS-5 and AWS environments during the Internal Variation and Behaviour experiments. Compared to the DAS-5, each experiment on AWS shows increased variation of RTT. Additionally, during the Behaviour experiment where each server was overloaded for a majority of the experimental duration, we observe an increase in the variation of RTT.

As a comparison, we show the Network experiment in Figure 15, where the players connected from an AWS node in California to another node in Paris. In this configuration, there was an increase of minimum to .14 seconds but not a significant amount of additional variation compared to the experiments on the AWS environment with no geographical distance.

[4]The PaperMC project seeks to improve over the performance of the Vanilla server via asynchronous multi-threading, and thus may suffer in environments with low vCPU count.

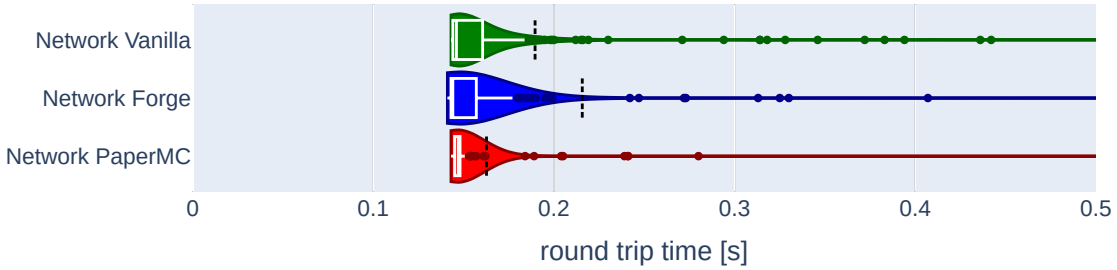


Figure 15: Round trip time violin plot for each server on AWS during the Network experiment. Black line depicts arithmetic mean.

Experiment	Minecraft-like Game	MB sent
Internal	Forge, Vanilla	602
Internal	PaperMC	426
Behaviour	Forge, Vanilla	319
Behaviour	PaperMC	275

Table 6: MB sent by each Minecraft-like game in the DAS-5 and AWS environments during the Internal Variation, Network and Behaviour experiments.

Table 6 shows the number of bytes by each Minecraft-like game in the AWS environment, showing that compared to the Internal variation experiment, during the Behaviour experiment in the AWS environment about 50% less bytes were sent by Forge and Vanilla, and around 40% less by PaperMC.

As during the Behaviour workload on AWS each server was overloaded and close to full utilization of the CPU, we propose that network bandwidth and RTT of Minecraft-like servers is dependant on CPU availability, rather than network latency. This observation is in line with prior findings, namely the 2019 Yardstick report which found that network traffic of Minecraft-like games is limited by CPU utilization [35].

6.6 MF5: Complex game worlds increase the performance requirement of MVEs.

In Figure 16 we compare the tick times of each server on AWS during the Internal Variation and World experiments, showing that the FutureCity world caused marginally more performance variation (around an 15ms increase to interquartile range) and a slight increase in median tick time (5ms on PaperMC and 10ms on Forge and Vanilla).

In the Internal Variation experiment the servers use the Baseline world and in the World experiment they use the community made “FutureCity” map (see Figure 7). The FutureCity map is significantly more complex than the Baseline world, being around 113MB [40] in size compared to the Baseline world of 5.4 MB.

Thus, as each server shows more performance variation in the same environment when using the FutureCity world, we conclude that complex environments increase the

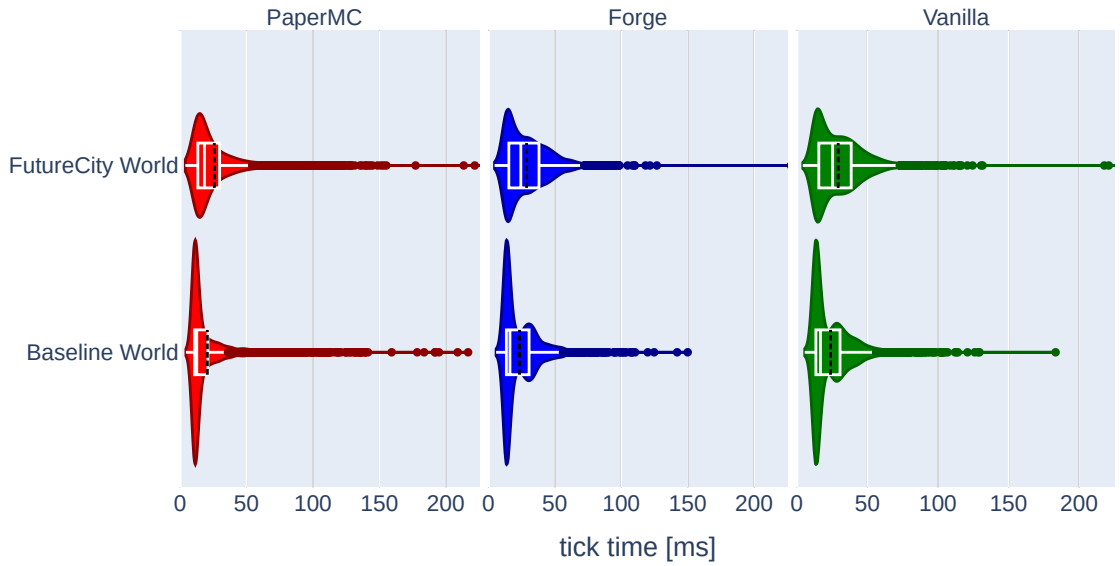


Figure 16: Tick time violin plot for each server on AWS when using the Baseline and FutureCity Worlds. Black line shows arithmetic mean.

performance requirements of the Minecraft-like games. This effect can be reasonably assumed to be caused by both the additional chunk data that must be serialized compared to the baseline world, as well as the presence of dynamic systems in the game world that require ongoing computation.

7 Discussion

In this section we outline the relevant prior research and how this thesis fits into that body of work, as well as the threats to validity.

7.1 Related Work

Research specific to benchmarking Minecraft-like games is limited. The main work within this niche being Van Der Sar, Donkervliet, and Iosup (ICPE, 2019) [35] wherein the Yardstick tool (extended in this thesis) was designed and used to benchmark the scalability of Minecraft-like games. While the Yardstick benchmark acknowledges the practice of using of cloud environments for the purposes of hosting Minecraft-like games, it does not incorporate into the design of the Yardstick tool or the benchmarking technique methods of measuring the impact of cloud environments. Accordingly, in our adaptation of the Yardstick benchmark design we add a deployment system suitable for cloud environments, as well as a control server and client system to facilitate distributed experiments and metric collection tools that are portable to many cloud environments and Minecraft-like games.

Work focusing on the use of serverless techniques as applied to Minecraft-like games is similarly limited, with a main work being Donkervliet, Trivedi, and Iosup (HotCloud-Perf, 2020) [7] which proposes a redesign of MVE technology to move certain sections of the MVE server to cloud environments. This work does not implement such a system

and thus also does not analyze the performance variability introduced from these cloud environments.

Performance variation in cloud environments is a well-studied topic, with the main research referenced in this work being Uta et al (USENIX, 2020) [34] which performed a wide-scale study of variability in cloud environments, with a focus on the network component. Additional work on variability in cloud environments includes Maricq et al (USENIX, 2018) [16], Taheri, Zomaya, and Kassler (Computing, 2017) [30], and Cao et al (USENIX, 2017) [5].

7.2 Limitations

In this work, we designed and implemented Meterstick, a benchmarking tool for Minecraft-like games operating in cloud environments. We then used Meterstick to measure the performance variability of popular Minecraft-like games in commercially available cloud computing services. We have ensured that the choice of Minecraft-like game and cloud environment are representative of real-world operation.

However, there is currently no publicly available real-world workloads of Minecraft-like games, such as average player count, typical player behaviours, or specific worlds used. For such decisions, we have assigned a best-guess value. In the case of player behaviour and game world, we used a domain coverage approach wherein we have one instance at a minimum realistic value (the bounded random player behavior and Baseline world) and another at a maximum (run away player behaviour and FutureCity world). Thus, we expect that in real-world operation most workloads should fall in the range between this minimum and maximum. For number of players, we chose 25 players informed by recommended values taken from community and official sources.

Additionally, though we have measured multiple quantitative metrics at both an application and systems levels as well as a quantitative approximation of subjective experience through round-trip time, we have no measurements of subjective values such as player experience. Thus, though we do observe values of server performance going above an overloaded point, we cannot conclude to what degree being in an overloaded state would effect player experience.

Finally, much as is the case for those operating real-world Minecraft-like games in a cloud environment, we do not have access to the internal allocation and scheduling data of the commercial cloud providers.

8 Conclusion

Minecraft-like games have societal importance stemming from general popularity as well as their value as tools for entertainment, education, and social causes. Minecraft-like games are increasingly being run in cloud environments in order to gain access to more computational power or network stability, but the performance effects of doing so is not well understood. Thus, this paper sought to answer three research questions: **RQ1**, on the design of a benchmarking tool for cloud-deployed Minecraft-like games, is addressed in Section 3 where we create a generalized design of Meterstick that satisfies our requirements for a fair and reproducible benchmark. **RQ2**, on the implementation of the design, is reported in Section 4 where we outline implementation details, challenges, and processes.

Finally, we answer **RQ3** on the performance variation of Minecraft-like games operated in real-world cloud environments with our main findings:

MF1 Minecraft-like games exhibit high amounts of performance variability in real-world cloud environments (Section 6.2).

MF2 Player behaviour significantly increases performance variation of Minecraft like games (Section 6.3).

MF3 Limited CPU resources effect each Minecraft-like game differently (Section 6.4).

MF4 Cloud environments increase variation in round trip time (RTT), especially when the Minecraft-like game is in an overloaded state (Section 6.5).

MF5 Complex game worlds marginally increase the performance requirement of MVEs (Section 6.6).

In future work, further experiments with more Minecraft-like games and commercial cloud providers would increase confidence in the overall trend of increase performance variation in cloud environments. Additionally, if the workload data from real-world Minecraft-like game servers can be obtained, then experiments could be made more representative through more advanced player behaviour models and accurate worlds.

9 Artifacts for Reproduction

We make all artifacts available as free open-access data, documentation, and open-source software.

Data: Available on Zenodo [9]
Software/Documentation: Available on GitHub [8]

References

[1] Amazon Web Services. Amazon ec2 t3 instances. `https://aws.amazon.com/ec2/instance-types/t3/`. Accessed: 2021-05-16.

[2] Autcraft. Autcraft - the first minecraft server for children with autism and their families. `www.autcraft.com`. Accessed: 2021-01-18.

[3] H. Bal, D. Epema, C. de Laat, R. van Nieuwpoort, J. Romein, F. Seinstra, C. Snoek, and H. Wijshoff. A medium-scale distributed system for computer science research: Infrastructure for the long term. *Computer*, 49(05):54–63, may 2016.

[4] BuildTheEarth. Build the earth - we are recreating the entire planet in minecraft. `https://buildtheearth.net/`. Accessed: 2021-01-18.

[5] Z. Cao, V. Tarasov, H. P. Raman, D. Hildebrand, and E. Zadok. On the performance variation in modern storage stacks. In *15th USENIX Conference on File and Storage Technologies (FAST 17)*, pages 329–344, Santa Clara, CA, Feb. 2017. USENIX Association.

[6] Curse Forge. All mods. https://www.curseforge.com/minecraft/mc-mods?filter-game-version=&filter-sort=5. Accessed: 2021-04-20.

[7] J. Donkervliet, A. Trivedi, and A. Iosup. Towards supporting millions of users in modifiable virtual environments by redesigning minecraft-like games as serverless systems. In *Proceedings of the 12th USENIX Workshop on Hot Topics in Cloud Computing*, pages 1–9. USENIX, Aug. 2020. 12th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2020, co-located with USENIX ATC 2020 ; Conference date: 13-07-2020 Through 14-07-2020.

[8] J. Eickhoff. Meterstick. https://github.com/JerritEic/Meterstick.

[9] J. Eickhoff. Meterstick data and plotting tools. https://doi.org/10.5281/zenodo.4964824, June 2021.

[10] Forge Development LLC. Minecraft forge. https://files.minecraftforge.net/, 2021. Accessed: 2021-02-10.

[11] G. Heiser. Systems benchmarking crimes. http://www.cse.unsw.edu.au/~Gernot/benchmarking-crimes.html#sign, 2019. Accessed: 2021-04-20.

[12] Hello Games. No man's sky. https://www.nomanssky.com/. Accessed: 2021-01-18.

[13] A. Iosup, L. Versluis, A. Trivedi, E. V. Eyk, L. Toader, V. van Beek, G. Frascaria, A. Musaafir, and S. Talluri. The atlarge vision on the design of distributed systems and ecosystems. *CoRR*, abs/1902.05416, 2019.

[14] R. Jain and R. JAIN. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. Wiley professional computing. Wiley, 1991.

[15] Keen Software House. Space engineers. https://www.spaceengineersgame.com/. Accessed: 2021-01-18.

[16] A. Maricq, D. Duplyakin, I. Jimenez, C. Maltzahn, R. Stutsman, and R. Ricci. Taming performance variability. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 409–425, Carlsbad, CA, Oct. 2018. USENIX Association.

[17] A. Megalios, R. Daly, and K. Burgess. MetaboCraft: building a Minecraft plugin for metabolomics. *Bioinformatics*, 34(15):2693–2694, 03 2018.

[18] Microsoft. Dv3 and dsv3-series. https://docs.microsoft.com/en-us/azure/virtual-machines/dv3-dsv3-series. Accessed: 2021-05-16.

[19] Minecraft Wiki. Server/requirements/dedicated. https://minecraft.fandom.com/wiki/Server/Requirements/Dedicated. Accessed: 2021-05-16.

[20] Mojang. Realms for java. https://www.minecraft.net/en-us/realms-for-java. Accessed: 2021-04-16.

[21] Mojang. Download the minecraft: Java edition server. https://www.minecraft.net/en-us/download/server, 2021. Accessed: 2021-04-16.

[22] Oracle. Monitoring and management using jmx technology. https://docs.oracle.com/javase/8/docs/technotes/guides/management/agent.html. Accessed: 2021-04-16.

[23] J. Ousterhout. Always measure one level deeper. *Commun. ACM*, 61(7):74–83, June 2018.

[24] Re-Logic. Terraria. https://terraria.org/. Accessed: 2021-01-18.

[25] Reporters Without Borders Germany. The uncensored library. https://uncensoredlibrary.com/en. Accessed: 2021-01-18.

[26] F. Richter. Amazon leads $130-billion cloud market. https://www.statista.com/chart/18819/worldwide-market-share-of-leading-cloud-infrastructure-service-providers/. Accessed: 2021-05-16.

[27] K. E. Ringland, L. Boyd, H. Faucett, A. L. Cullen, and G. R. Hayes. Making in minecraft: A means of self-expression for youth with autism. In *Proceedings of the 2017 Conference on Interaction Design and Children*, IDC '17, page 340–345, New York, NY, USA, 2017. Association for Computing Machinery.

[28] G. Rodola. psutil 5.8.0. https://pypi.org/project/psutil/, 2020. Accessed: 2021-04-10.

[29] System Era Softworks. Astroneer - a game of aerospace industry and interplanetary exploration. https://astroneer.space/. Accessed: 2021-01-18.

[30] J. Taheri, A. Y. Zomaya, and A. Kassler. vmbbprofiler: a black-box profiling approach to quantify sensitivity of virtual machines to shared cloud resources. In *Computing*, pages 1149–1177, Aug. 2017.

[31] The Paper MC Team. Papermc - the high performance fork. https://papermc.io/, 2021. Accessed: 2021-02-10.

[32] S. Tolbert. Minecraft crosses 131 million monthly active users. https://www.windowscentral.com/minecraft-crosses-131-million-monthly-active-users, 2020. Accessed: 2021-01-18.

[33] S. Tolbert. Minecraft passes 200 million copies sold. https://www.windowscentral.com/minecraft-passes-200-million-copies-sold, 2020. Accessed: 2021-01-18.

[34] A. Uta, A. Custura, D. Duplyakin, I. Jimenez, J. Rellermeyer, C. Maltzahn, R. Ricci, and A. Iosup. Is big data performance reproducible in modern cloud networks? In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 513–527, Santa Clara, CA, Feb. 2020. USENIX Association.

[35] J. Van Der Sar, J. Donkervliet, and A. Iosup. Yardstick: A benchmark for minecraft-like services. In *ICPE 2019 - Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 242–252. Association for Computing Machinery, Inc, Apr. 2019. 10th ACM/SPEC International Conference on Performance Engineering, ICPE 2019 ; Conference date: 07-04-2019 Through 11-04-2019.

[36] C. Waters and B. Trevethan. How minecraft realms moved compute+storage from aws to azure. `https://developer.microsoft.com/en-us/games/blog/how-minecraft-realms-moved-compute-storage-from-aws-to-azure`. Accessed: 2021-05-16.

[37] R. Weicker. Benchmarking. In M. C. Calzarossa and S. Tucci, editors, *Performance Evaluation of Complex Systems: Techniques and Tools*, pages 179–207, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[38] W. Witkowski. Videogames are a bigger industry than movies and North American sports combined, thanks to the pandemic. `https://www.marketwatch.com/story/videogames-are-a-bigger-industry-than-sports-and-movies-combined-thanks-to-the-pandemic-11608654990`, 2020. Accessed: 2021-04-16.

[39] A. Yahyavi and B. Kemme. Peer-to-peer architectures for massively multiplayer online games: A survey. *ACM Comput. Surv.*, 46(1), July 2013.

[40] Zeemo. Future City. `https://www.minecraftmaps.com/city-maps/future-city`, 2017. Accessed: 2021-01-17.

A Hardware Used in Cloud Deployed Minecraft-like Games

Service	RAM	CPU	Notes
Hostinger	3GB	3vCPU	CPU GHz not listed
Nodecraft	4GB	3.8GHz	Number of vCPU not listed
Apex Hosting	4GB	3.9GHz	Number of vCPU not listed
ScalaCube	3GB	2x3.4GHz	
GGServers	4GB	3.2GHz	Number of vCPU not listed
Server.pro	4GB	2x2.4GHz	
BisectHosting	4GB	3.4Ghz	Number of vCPU not listed
Shockbyte	4GB	4.0Ghz	Number of vCPU not listed
CubedHost	2.5GB	4.5Ghz	Number of vCPU not listed
ServerMiner	3GB	4GHz	Number of vCPU not listed
Akliz	4GB	3.4GHz	Number of vCPU not listed
RamShard	2GB	4GHz	Number of vCPU not listed
MCProHosting	2GB	NA	CPU info not available
GTXGaming	3GB	1x3.8GHz	
StickyPiston	2.5GB	NA	CPU info not available
HostHavoc	4GB	4GHz	Number of vCPU not listed
Skynode	4GB	2x3.6GHz	
Ferox Hosting	4GB	NA	CPU info not available
Aquatis	4GB	4.2Ghz	Number of vCPU not listed
PebbleHost	3GB	3.7Ghz	Number of vCPU not listed
MelonCube	4GB	3.4Ghz	Number of vCPU not listed

Table 7: Recommended plans from services that offer Minecraft hosting.

Cloud hosting specifically tailored for Minecraft servers is a common service, in Table 7 we list a sampling of popular Minecraft hosting services and the hardware specifications that they report. The data shown is taken from hosting plans that are marked as "recommended" or comparable to this recommended plan on other services, if none were marked as such. We see a lack of transparency in hosting configuration, where many services list the hardware used in their data centers but do not report the use of virtual machines partitioning, hypervisors or other allocation system details.

The hardware requirements of Minecraft that each service reports vary greatly, with some saying that 4GB of RAM is suitable for many players and mods, and others reporting that it is enough for only 20 players.

Aside from these services that are tailored to Minecraft-like games, there is also the option to use general purpose cloud computing such as offered by Amazon Web Services, Microsoft Azure and Google Cloud. For each of these services, there are either official guidelines for operating Minecraft-like games or there are unofficial recommendations, listed in Table 8. These recommendations are not uniform, suggesting there is a lack of consensus on the hardware requirements of Minecraft-like games.

Notably, very few of the services or recommendations listed meet hardware recommendations for dedicated servers with more than 20 players as published by the Minecraft

Service	Source	RAM	CPU
AWS	Official	1GB	1vCPU
AWS	Unofficial	7.5GB	2vCPU
Azure	Official	4GB	2vCPU
Azure	Unofficial	3.5GB	1vCPU
Azure	Unofficial	2GB	1vCPU
Google Cloud	Unofficial	1.7GB	1vCPU

Table 8: Recommended plans for general purpose cloud computing providers.

Wiki: 6GB RAM and an modern Intel or Ryzen CPU (8 to 12vCPU). However, this wiki also states that single-core performance is more important to Minecraft than many threads [19].

From these sources we find that 4GB RAM and 2vCPU is most often recommended. Thus, in our experiments we utilize nodes with 8GB RAM and 2vCPU, with the Minecraft-like games limited to 4GB of RAM through the JVM "-Xmx" argument. This configuration is chosen both to ensure our metric sampling tools have sufficient memory and because none of the services listed specify if the RAM requirement for the Minecraft server is a hardware maximum (that would include the needs of the OS, JVM, and networking programs) or a software limit (such as specified by the JVM).