Vrije Universiteit Amsterdam

Honours Program, Project Report

Evaluating the Performance of Player-Agnostic Systems of Minecraft-Like Games using Environment-based Workloads

Author: Jerrit Eickhoff (2641983)

1st supervisor: prof. dr. ir. Alexandru Iosup
daily supervisor: ir. Jesse Donkervliet

A report submitted in fulfillment of the requirements for the Honours Program, which is an excellence annotation to the VU Bachelor of Science degree in Computer Science/Artificial Intelligence/Information Sciences
version 1.0

February 23, 2021

The Project Card

Research question(s):

RQ1 What are the characteristics of player-agnostic workloads?

RQ2 How to evaluate the performance of Minecraft-like games under player-agnostic workloads?

RQ3 What is the performance of Minecraft-like games when running player-agnostic workloads?

Research method(s):

M1 Design of a workload abstraction and of a full benchmark.

M2 Experimental research in computer systems, method synthesized by Raj Jain.

M3 Performance analysis using quantitative data from real-world experiments.

Research approach:

A1 To answer **RQ1**, we apply the iterative AtLarge design methodology (**M1**) to create an abstraction of Player Agnostic workloads.

A2 For **RQ2** we use the experimental research methodology (**M2**) when designing our experiment, which guides our choice of metrics, workloads, and benchmarks to compare against as well as our experimental configuration of repetition and averaging.

A3 We answer **RQ3** by creating and using a benchmark (**M3**). The benchmark focuses on MVEs in real world settings that evaluates appropriate quantitative system and application level metrics in order to avoid relating macro-level performance to a singular micro-level metric.

Main results (main findings, MFs) and contribution:

C1 Defined a new class of MVE scalability problems, that of managing player-agnostic workloads (**RQ1**), and an operational model based on these workloads.

C2 Defined a benchmark to evaluate the effect of player-agnostic workloads on MVE performance through real-world experiments (**RQ2**).

C3 Conducted extensive real-world experiments on three common implementations of Minecraft-like games, including the most popular today, and analyzed the results.

MF1 Player-agnostic workloads can significantly affect the performance of Minecraft-Like games.

MF2 The Minecraft-like services we studied, including the commercial ("vanilla") Minecraft, are poorly parallelized, which impacts negatively their ability to run player-agnostic workloads.

MF3 Processing the state of in-game entities is computationally expensive for player-agnostic workloads, and can cause server overloads.

Contents

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. These games are characterized by the utilization of Modifiable Virtual Environments (MVEs). The popularity of games using MVEs has created demand for high-performance hosted environments running multiplayer game services. Despite the demand, prior research has shown that current MVE server technology does not scale well to large amounts of concurrent players. Additionally, little research has focused upon Player Agnostic Workloads, a unique performance pattern of MVEs that scales separately from player count and behaviour. MVEs are both infinitely customizable and contain dynamic behavior, and thus have the potential for high complexity. In short, parts of a modifiable world can become complex systems in their own right. In this work, we define a model of common player-agnostic systems, and evaluate the performance of MVE services when using game environment workloads that contain example player-agnostic workloads. Our findings show that Minecraft-like service performance is highly reliant on the workloads inside of the MVE, such that specific configurations of the game environment can cause service quality degradation. **Keywords** Scalability; Minecraft; workload; online-gaming.

1 Introduction

The global gaming market generated revenue in access of $159 billion in 2020, a value expected only to rise in the coming years [31]. Minecraft, as the second most best-selling video game of all time [28, 40], accounts for a significant amount of the global revenue, having sold 88 million copies since 2017 [48], and having reported more than 131 million players per month [47]. Minecraft provides a platform not only for individual entertainment, but community movements with a broad spectrum of causes, from seeking to build the entirety of Earth [4], visualization tools for metabolic pathways [26], to creating safe spaces for autistic players [1, 37], and facilitating an in-game haven for press freedom to fight against real world censorship [36]. However, the potential of Minecraft-like games is bound by the limitations of their multiplayer services, which support a maximum player count in the low hundreds. This lack of scalability makes it difficult for large communities or events to utilize Minecraft-like games as a medium. Prior work in measuring and increasing the scalability of Minecraft-like services is limited, and has focused on computational requirements that scale by player count [6, 50, 33]. This approach does not take into account the non-trivial computational workload of the environment in Minecraft-like games. In this work, we define a model for these *player-agnostic workloads* in Minecraft-like environments, then design and perform experiments to measure the performance impact of these workloads.

Minecraft's most unique feature is the use of a *Modifiable Virtual Environment* (MVE), a real-time, dynamic, multi-user environment which allows players to freely destroy and create components with nearly unbounded scale. Minecraft's aforementioned success has popularized the use of MVEs in a variety of newer Minecraft-Like games, such as No Man's Sky [13], Astroneer [42], Terraria [34], and Space Engineers [18]. While MVEs provide a high degree of player freedom, prior research has found that current MVE technology suffers from poor scalability: Minecraft-like servers can only concurrently manage maximum player counts in the low hundreds, often experience performance degradation before that maximum, and do not interact with other server instances [50].

Video game scalability is not a new problem, and many techniques have been developed to increase concurrent player count. A majority of these approaches were developed for video games that do not utilize MVEs, and have only limited applicability. Approaches that are relevant to MVEs are restricted to techniques that minimize workloads that scale with player count, such as world partitioning [15], interest management [3, 23], and inconsistency management [45, 9]. However, Minecraft-like games do not only process player operations, they also compute environment actions not directly tied to players.

In this work, we define a novel model of player-agnostic workloads to describe these workloads that are characteristic to MVEs and occur as a result of complex environment dynamics, rather than by player count. The computational requirements of these environment-based computations does not only affect player count scalability, but workload scheduling, environment complexity limits, and service guarantees for web hosting platforms. Thus, we pose the following main research question: *What is the performance impact of player-agnostic workloads in MVEs?* We then divide this question into three research questions:

- **RQ1 (Section 3):** What are the characteristics of player-agnostic workloads?
  A core activity for players of Minecraft-like games is the construction of structures that function as Player-Agnostic workloads. Because MVEs are uniquely configurable into an infinite variety of layouts it is difficult to identify a finite set of relevant Player-Agnostic workloads. Instead, identifying what kinds of features within these configurations have a performance impact requires defining a model to separate them into classes. This is non-trivial as different Minecraft-like games do not have identical features, thus the model must differentiate by function, rather then by form.

- **RQ2 (Section 4):** How to evaluate the performance of Minecraft-like games under player-agnostic workloads?
  Minecraft-like games, like all real-time systems, have strict performance requirements to prevent service degradation, and evaluating this performance is a crucial part of operating Minecraft-like games as a service. In order to measure the performance of a Minecraft-like game when operating a player-agnostic workload, it is necessary to design an experimental method. This also requires defining performance in the context of a Minecraft-like service. This is difficult as the experiment must be as close to a real world setting as possible in order to maintain validity, but still be repeatable.

- **RQ3 (Section 5):** What is the performance of Minecraft-like games when running player-agnostic workloads?
  Understanding how Player-Agnostic workloads effect the performance of Minecraft-like services is important in mitigating performance degradation and designing new scalability techniques. The experimental measuring of Minecraft-like game performance requires a set of player-agnostic workloads for the Minecraft-like services to operate. Thus, it is necessary to find a set of player agnostic workloads that are both representative of the classes established in the workload model, and can be reason-

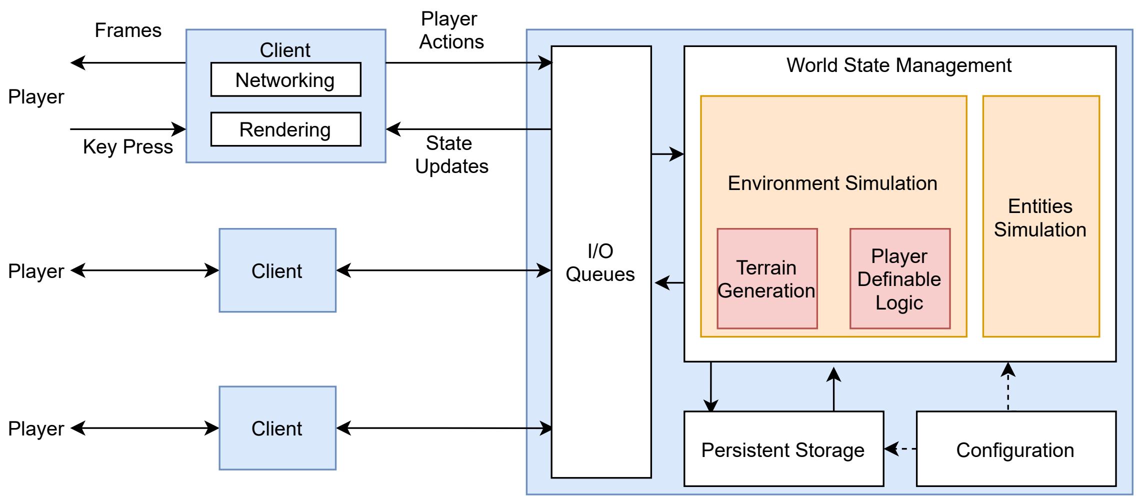

Figure 1: Overview of a Minecraft-like MVE system.

ably assumed to be realistic. Additionally, the metrics measured by the experiment must be properly analyzed to discern the proportion of performance that is due to changes in environment workload.

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

- **M1:** To answer **RQ1** we apply the iterative AtLarge design methodology to create an abstraction of Player Agnostic workloads [19].
- **M2:** For **RQ2** we use the experimental research methodology when designing our experiment, which guides our choice of metrics, workloads, and benchmarks to compare against as well as our experimental configuration of repetition and averaging [32, 17].
- **M3:** We answer **RQ3** by creating a benchmark of MVEs in real world settings that evaluates appropriate quantitative system and application level metrics in order to avoid relating macro-level performance to a singular micro-level metric [17].

2 Background

In this section we define important terminology and give an overview of Minecraft-like game functionality.

2.1 Minecraft-like Games

We consider a game *Minecraft-like* if, at a minimum, it operates using a Modifiable-Virtual Environment (MVE). MVEs allow players to concurrently modify the world in creative ways, creating player interactions not possible in other games. Figure 1 shows the design of a Minecraft-like MVE system. Common, but not ubiquitous features of Minecraft-like games include a server-client model for multiplayer service, non-player characters that inhabit the environment, and some form of player definable logic.

2.2 Player-Agnostic Workloads

A *Player-Agnostic Workload* is any server-side workload in a Minecraft-like game that scales independently from the number of connected players. This is contrary to typical video game server workloads that scale through the need to send player actions to all other connected players. It is important to acknowledge that player agnostic workloads can result from player modifications within the modifiable environment. Thus, more players will generally result in an increase in the computational strain from player-agnostic workloads, but crucially only one player[1] is required to create a player-agnostic workload of any complexity.

In Section 3 we give a model that classifies types of Player Agnostic Workloads based upon their relation to features of the MVE.

2.3 Entities

Minecraft-like games often include *entities* that interact with the environment, making the world more interesting to players. Within MVEs, a given game feature is considered an entity if it exists within the game world but is not a player or a part of the modifiable terrain. Within this definition, there are a few main types of entities: *driven entities* exhibit some form of decision making and movement ability (often referred to as "*mobs*"), and *non-driven entities*, which generally represent an in-game resource or provide some functional or visual effect. Notably, many of these entities have the ability to alter the environment of their own accord. For example, in Minecraft itself, Creeper entities can destroy blocks, Endermen entities can move blocks to different locations, and Villagers tend to animals and harvest crops. Aside from animal and people-like entities, there are a wide variety of non-driven entities such as dropped items, vehicles, and weapons; some of which can be persistent, such as chests.

2.4 Player Definable Logic

Many Minecraft-like games have a feature enabling *Player Definable Logic* that extends the game environment by adding some form of custom circuitry into the game calculations. The primary example of this is Redstone [35] in Minecraft, but other examples include Conveyor [7] systems in Satisfactory, Wire [52] in Terraria, Circuits [5] in Factorio, Modules [27] in Astroneers, and Logic [25] in No Man's Sky. Importantly, there is no hard limit to the computation complexity that player built circuitry can achieve. Many of these systems are even Turing complete [51], allowing, for example, functional computer systems to be built within the game environment.

A *Farm* refers to a special type of Player Definable Logic, wherein a player creates a system to automate the collection of some in-game resource. Farms are perhaps the most common form of Player Definable Logic, and a core feature within many Minecraft-Like games. Often, a Farm will utilize the functioning of driven and non-driven entities in order to facilitate resource collection.

[1]Depending on how a Minecraft-like game is configured, it is possible for the server to continue environment computation when no players are connected, thus technically no players are strictly required.

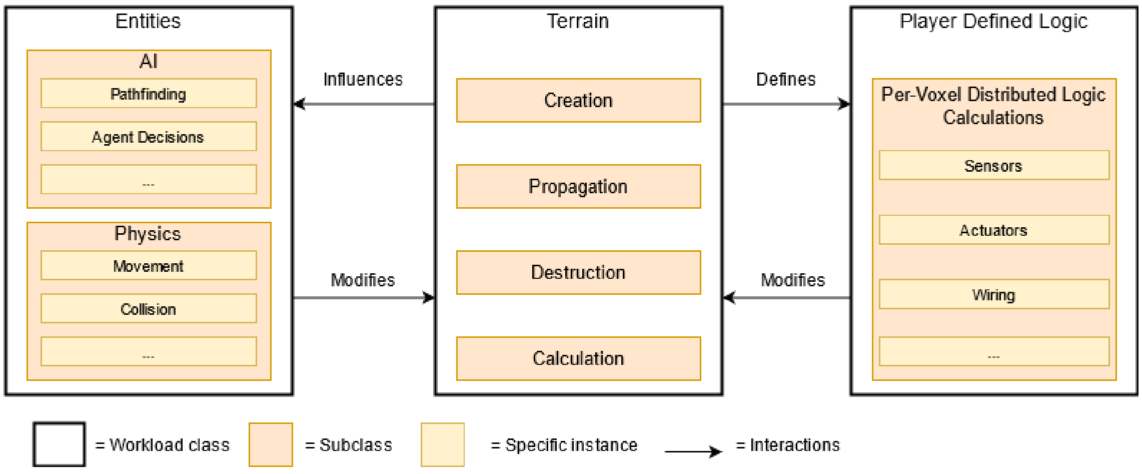

Figure 2: Player Agnostic Workloads model with classes and sub-classes.

3 Workload Model

In this section we propose a model that separates types of player-agnostic workloads into three overarching classes with shared traits. Figure 2 depicts this model. We design this model with a focus upon Minecraft, and these workloads are supported by all Minecraft-like games that utilize the Minecraft protocol. Anecdotally this model will generalize well to other Minecraft-like games due to the similarity in MVE structure between games.

3.1 Terrain

By definition, all MVEs contain modifiable environments. MVEs have dynamic behaviour that controls or changes environment modifications, such as physics calculations, fire, water, crop growth effects, or any other world-based calculations that modifies the world indirectly from the player. In Figure 3 we give examples inherent to MVEs. These types are Terrain Creation, referring to adding novel structures to the environment, Terrain Propagation, by which properties are spread between areas of the environment (in the example, the property is "on fire"), Terrain Destruction, for any action that removes sections from the environment, and Terrain Calculations, which uses the environment to compute some value (in the example, light level).

While there are comparable features in games that do not use MVE systems, terrain workloads in MVEs function differently due to how MVEs are structured: since the environment is fully modifiable, and often procedurally generated, no assumptions can be made about world state or configuration. Thus all terrain actions must be calculated as a function of current neighboring state. In Minecraft, for instance, terrain actions require a data structure containing the state of all neighboring voxels (*blocks*). Thus, world modification is computationally expensive as each modification action requires retrieving, calculating and updating the state of neighboring blocks, and is further complicated when concurrent world actions access the same blocks. Within the group of Terrain workloads, notable examples that are particularly computationally complex are explosives, lighting calculations, and physics simulations, as these tend to be spontaneous, hard to predict,

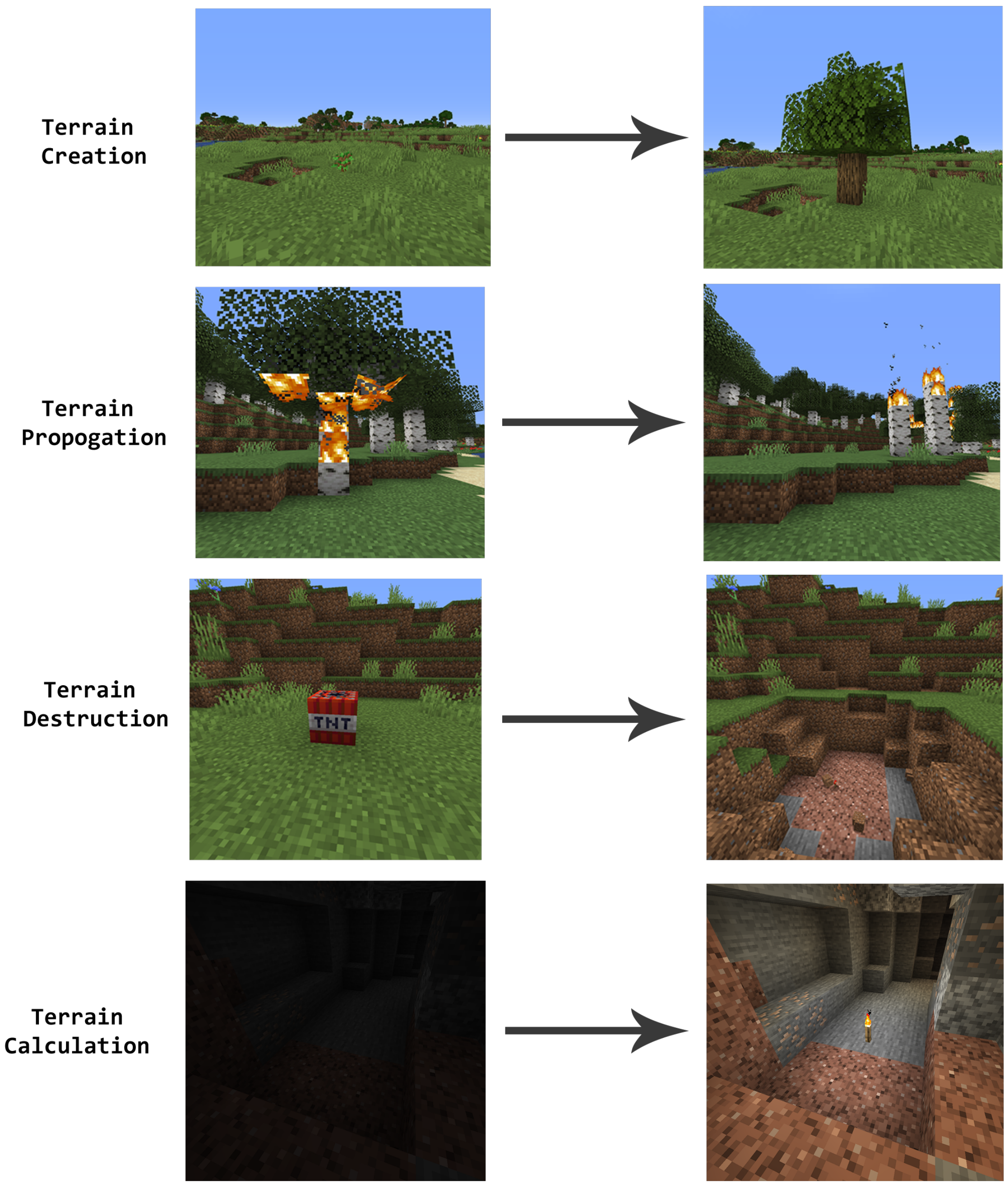

Figure 3: Classes of terrain modification and examples from Minecraft.

and use a high volume of environment state updates.

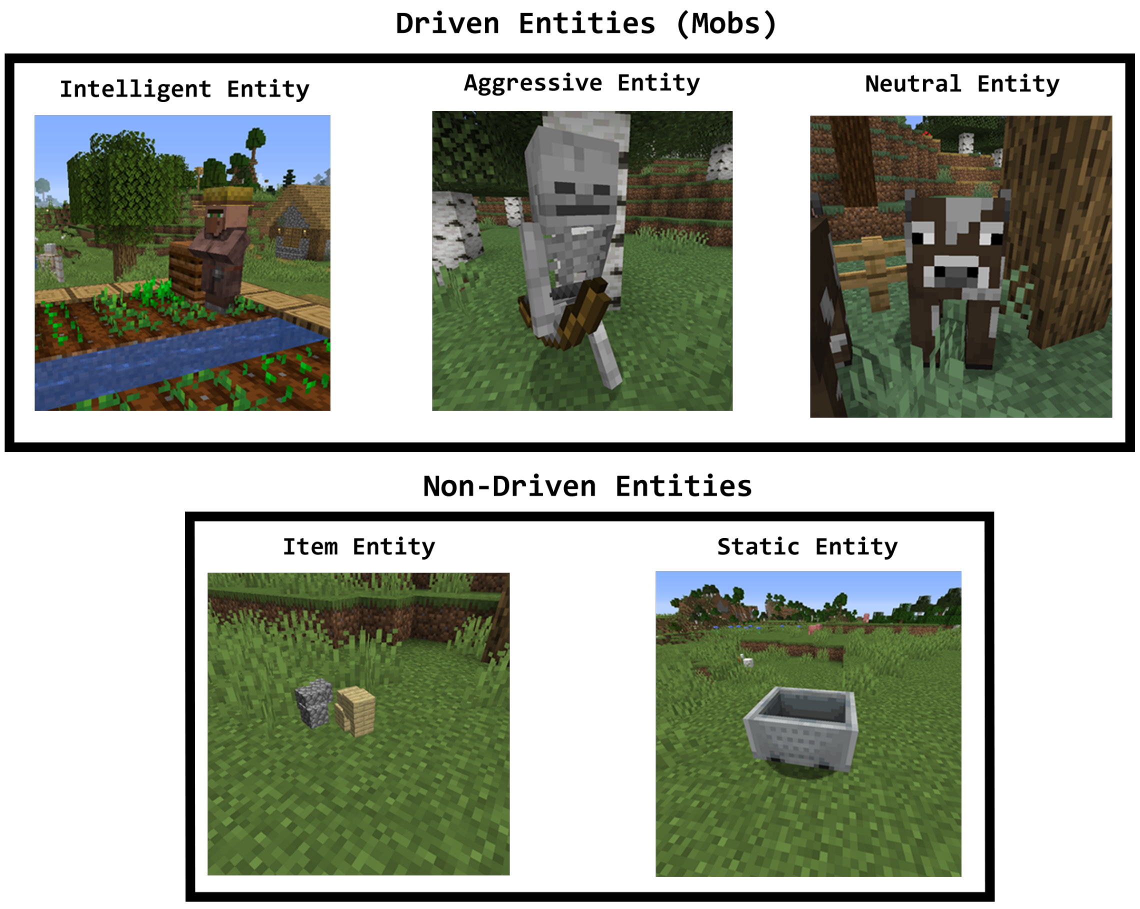

Figure 4: Classes of entities and examples from Minecraft.

3.2 Entities

A prominent feature of Minecraft-like game is the presence of non-player entities within the game world. In general, such entities can either be driven, and have a form of AI that decides its actions, or be non-driven, and still be able to interact with the world but have no advanced decision making capabilities.

Figure 4 gives an overview of entities types, with *Intelligent*, *Aggressive* and *Neutral* entities having some form of AI, and Item and Static entities having none. Intelligent entities behavior is complex and often revolves around interacting with the environment, with the example given being a Farmer Villager, which will plant and harvest crops of their own accord. Aggressive Entity actions are not decided by environmental factors but instead by the player, as their main goal is follow and harm the player. Neutral Entities actions are not heavily influenced by either environment or player, but instead largely random movements around the game world. However, all driven entities require environment state in order to make decisions and for pathfinding, increasing the computational need of the MVE.

Static entities have no AI, but can be interacted with, moved or used by the player. Item entities are a common feature of MVEs in which amounts and types of resources are

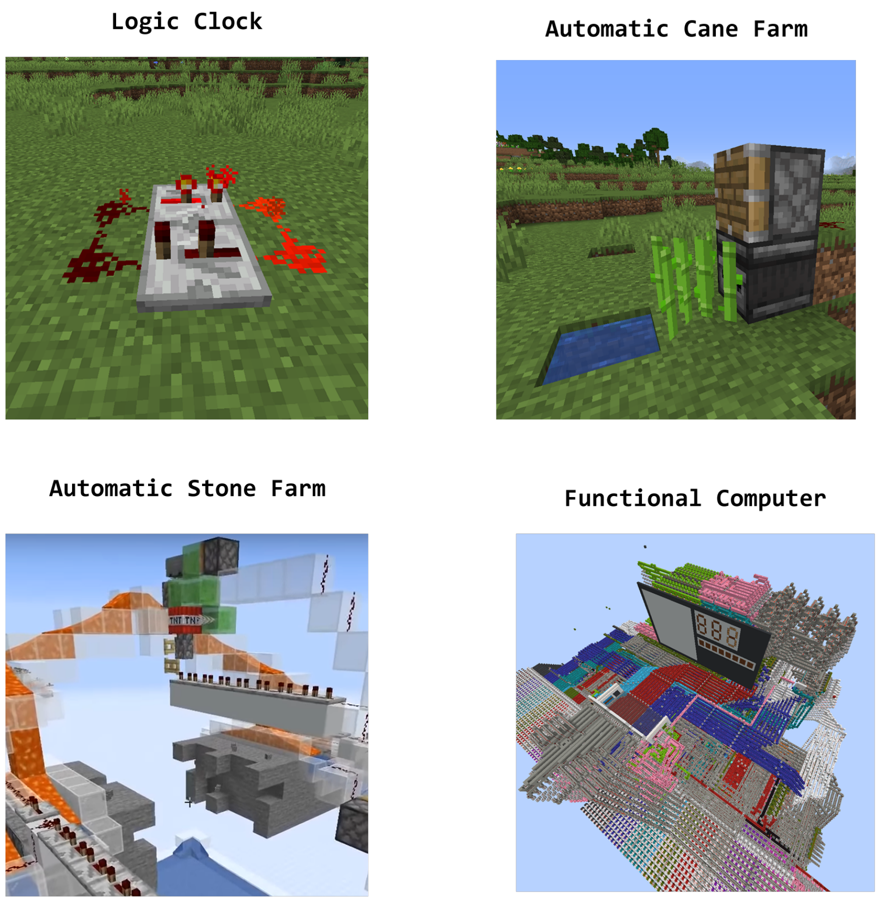

Figure 5: Structures utilizing player defined logic.

represented in the environment by a smaller icon of the object. While static entities have no decision capability, they still require computation for collision and movement.

Much like Terrain workloads, there are similar features to all of these entity types in games that aren't Minecraft-Like, but due to the unique nature of MVEs they do not act the same way. For instance, pathfinding is a common driven entity calculation, but because MVE terrain is modifiable there are no "checkpoint" systems to pre-bake paths as is done in games without MVEs. Other examples of strenuous workloads within this class are entity collision and item entity movement, such as through water streams or other "conveyor-belt" systems.

3.3 Player-defined logic

Minecraft-like games give players the ability to create custom circuitry by building structures that behave like in-world sensors, wires, and actuators. These components can be chained together to make an infinite variety of devices each with unique behaviour. Instead of classifying all possible structure types, example structures are described that are known to be common usages of player defined logic.

Figure 5 contains four important examples of player-defined devices that have equivalencies in other Minecraft-Like games. The Logic Clock is an example that is close to traditional circuitry, creating a pulse with a fixed frequency. The two farm examples represent the most typical usage of player-defined logic; their purpose is to automate resource collection for the player by modifying the terrain to generate and subsequently harvest resources.

Player defined logic in this form is unique to MVEs as it is integrated into the world environment, rather than being explicitly defined. This property is also why player defined logic is central to the concept of player agnostic workloads: it is prohibitively difficult to predict the behaviour of a given player definable logic system as its actions rely both on measuring and modifying the state of the surrounding environment. Additionally, player-definable logic can act randomly by reacting to random state change events in the environment.

Reliance on and ability to change neighboring state as well as having capability to be random means that player definable logic can exhibit high complexity. Generally this is a positive feature of the MVE as it allows for designs with high degrees of intricacy and expressiveness, such as the given example of a functional computer made entirely of Redstone. However, this intricacy comes with a performance cost.

The computation complexity of player defined logic has social implications: using player defined logic it is possible to create "*lag-machines*," a form of denial of service attack, wherein a malicious player creates structures designed to be computationally demanding intended to overload or crash the server.

In short, this category contains any player-built logic system that utilizes per-voxel calculations, but important examples are farm systems, as they are both computationally expensive and extremely common in all Minecraft-like games, as well as lag machines, as they represent the extreme case of complexity.

4 Experiment Design

In this section, we define a method to measure the performance impact of each Player-Agnostic workload: first we create a set of prototypical Minecraft worlds that contain experimental workloads corresponding to those listed in Table 1, then start a server with one of these worlds using the default recommended server configurations. Finally we connect a computer controlled player to the server, and record performance metrics for fifty seconds. We repeat this process three times for each server/workload combination, and average the results across iterations.

4.1 Choice of Workload

We decide upon specific types of Player Agnostic workloads based on real world applicability for all Minecraft-like games and to provide archetypal examples of the classes within the Workload Model introduced in Section 3.

Because this experiment specifically measures workload impact within a Minecraft-protocol server, workloads are chosen that were popular and common within the Minecraft community. However, each workload is anecdotally expected to be common within other Minecraft-like games as they utilize the core, unique features of Modifiable-Virtual Environments. Additionally, we use a control world to provide a baseline to compare the Player

World	**Properties**
Control	Default generated world
TNT	Large chunk of TNT
Resource Farms	Many community-made resource farms
Lag Machine	Complex Redstone device that stress-tests server

Table 1: List of workload worlds used in experiments.

Agnostic Workload performance against. An overview of workloads is listed in Table 1.

4.1.1 Control

A default generated Minecraft world[2], with no modifications. The player spawns in a plains biome, with sparse trees and a fair amount of passive entities spawning nearby. This world represents a minimum baseline in workload while still being a realistic use case. The measured results of this workload are used as a workload-level benchmark to compare the other workloads against.

4.1.2 TNT

A default Minecraft world, this time in a forest biome with more trees, where a 16 by 16 by 14 area of TNT Blocks has been added and rigged to explode when a player joins the server. High volume world modifications that are difficult to predict are a core problem within the Player Agnostic Workload model (Section 3), specifically within Terrain modifications. We chose TNT as it does exactly this, and activating large amounts of it is a popular activity, often to test the performance limits of a server.

This phenomenon can be seen in a plethora of community-made content. For example, a video made in 2018 by Minecraft YouTube creator DudeItsRocky with 21 Million views shows only an in-game explosion of thousands of TNT blocks [10]. In Minecraft, TNT that has been activated is a form of non-driven entity that can collided with and moved by other entities. Thus a TNT chain reaction requires spawning and calculating the movement of many entities, adding additional computational strain.

4.1.3 Resource Farms

This world spawns the player on a coastal forest, that contains a variety of popular Minecraft community designed Resource Farms. In particular, it contains twelve Aggressive Mob Farms, that automatically spawn and harvest enemy entities, four Cobblestone Farms, that utilize interactions between lava and water liquid blocks to create Cobblestone and harvests it, and four Kelp Farms, that detect crop growth and harvest only when it is ready. Alongside the Farms is a system of item entity ”pipes” that use water movement to bring the harvested resources to a central sorting and storage system.

We chose these farms based upon the popularity of their creators and the video showcases of the farm designs: Youtube users gnembon (1.7 Million views) [16], Shulkercraft

[2]random seed: -3699908958998125117

(1.3 Million views) [39], Mumbo Jumbo (2.5 Million views) [21], and Mysticat (800 Thousand views) [30] respectively, as this is assumed to be correlated with the amount these farms are used through the community.

Notably, the Mob and Cobblestone farm utilize a in-environment clock, similar to the one shown in Section 3.3, the pulses of which control their operation. Additionally, these farms rely on the actions of entity in their functioning. The Mob farm relies on spawning as many mobs as possible and using collisions and pathfinding to push them over a cliff. The Cobblestone farm continuously activates TNT, which while activated is an entity. Similarly, when mobs are killed or blocks destroyed, they drop item resources which are represented as passive entities. These item drops are then transported through terrain physics, using water streams.

4.1.4 Lag Machine

The lag machine world spawns the player on a small island, surrounded by water with few entities, where a large lag machine structure has been built. Whereas the previous workloads are typical examples of workloads than can be expected to happen during the normal operation of a Server, this workload is an extreme, and likely rare, case. Lag Machines are examples of Player Definable Logic wherein a malicious player, seeking to crash or reduce the performance of a server, creates a structure in game designed to require high computational power. This form of Denial of Service attack relies on the existence and impact of Player Agnostic Workloads.

We chose the specific design of the Lag Machine used in this workload from Youtube user Timeam (52 thousand subscribers) [46], as it a purely Redstone design, rather than relying on implementation-specific entity calculation exploits that may not have equivalencies in other Minecraft-like games.

Importantly, Redstone calculations such as the ones that this lag machine is based on are generally non-malicious, being both common and useful in normal operation of a Minecraft server, for instance forming the basis for impressive examples of Player Defined Logic such as the operational Redstone Computer [22].

4.2 Systems Under Test

To show that the performance impact of a given player-agnostic workload is not due only to the implementation details of a given server, we performed each experiment using three different Minecraft server applications. We select three Minecraft-like games: Minecraft, Forge, and PaperMC. The first is the official, "*Vanilla*," Minecraft server [29], chosen as it represents the default experience that most users and hosts utilizing Minecraft servers employ.

The second is the Forge server, which is an extended version of the Vanilla server that adds support for community made modifications ("*mods*"), and is the most popular choice for operating such modified servers [24]. Of the top 50 most downloaded Minecraft mods, only 5 are not Forge-exclusive, and of those 5, only 1 is not available on Forge [12].

The third is the PaperMC server, which we chose as it is marketed as an open source, high-performance alternative to the Vanilla server [44]. While the PaperMC project provides no quantitative figures on what performance gain can be expected compared to the Vanilla server, it does provide a list of optimizations, including extensive changes to threading models and environment processing.

We expect that the performance of each server will be negatively impacted by the workloads tested, albeit not to the same degree or in equivalent ways. Since each server alternative is configured for different purposes, their performance should differ when operating different workloads.

4.3 Experiment Tool

To facilitate performance analysis, we use and extend Yardstick [50], a benchmark for Minecraft-like games, to connect a bot that moves around the game world based on a player-behaviour model trained in Second-Life. This ensures that the server being analysed receives realistic player input over the course of the experiment.

4.4 Metrics Collected

We adapt and run the experiment design on the DAS-5, a distributed super-computer for academic and educational use [2]. In each experiment, a DAS-5 node runs the Minecraft server while another node connects to it using the Yardstick tool.

On the node that runs the server a Python script using a system performance monitoring library Psutil [38] as well as a plugin for Java JMX connections [14] logs CPU utilization percent, system packet I/O counts, and server *tick times*.

Tick time is an internal Minecraft measurement of the duration of one iteration of a game loop, with 50ms being the accepted maximum before noticeable performance issues (*server overload*). The total CPU utilization is measured as a sum of percentages for each CPU the server process is running on. To provide insight into network usage, the node running Yardstick logs each message it receives from the server.

Additionally, the Minecraft internal debugging logger was used to gain insight into what sections of the MVE took the most fraction of tick time, and the CPU analysis tool perf-map-agent [20] is used to log the thread and stack activity of the CPU over the course of the experiment.

5 Experiment Results

In this section we discuss the main findings based upon server performance data collected from the Vanilla, PaperMC and Forge servers when running the workloads listed in Table 1. Our main findings are:

- **MF1:** Player-Agnostic Workloads can significantly affect the performance of Minecraft-Like games (Section 5.1). Complex sections of MVEs are capable of overloading modern MVE server technology by up to 20 times the overloaded threshold, and there are large performance differences between Minecraft-like services. Of those tested, PaperMC performed the best while Forge performed worst.

- **MF2:** The Minecraft-like services we studied, including the commercial ("vanilla") Minecraft, are poorly parallelized (Section 5.2). This is evidenced by the trend of low CPU utilization during periods with high tick times. Even during high workload periods, none of the tested services utilized more than 20% of available CPU resources.

- **MF3:** Processing the state of entities is computationally expensive and can cause server overloads (Section 5.3). We find that 95% of all server messages are Entity related, and that 70% of non-idle tick time is Entity calculations.

5.1 MF1: Player Agnostic Workloads can significantly affect the performance of Minecraft-like games.

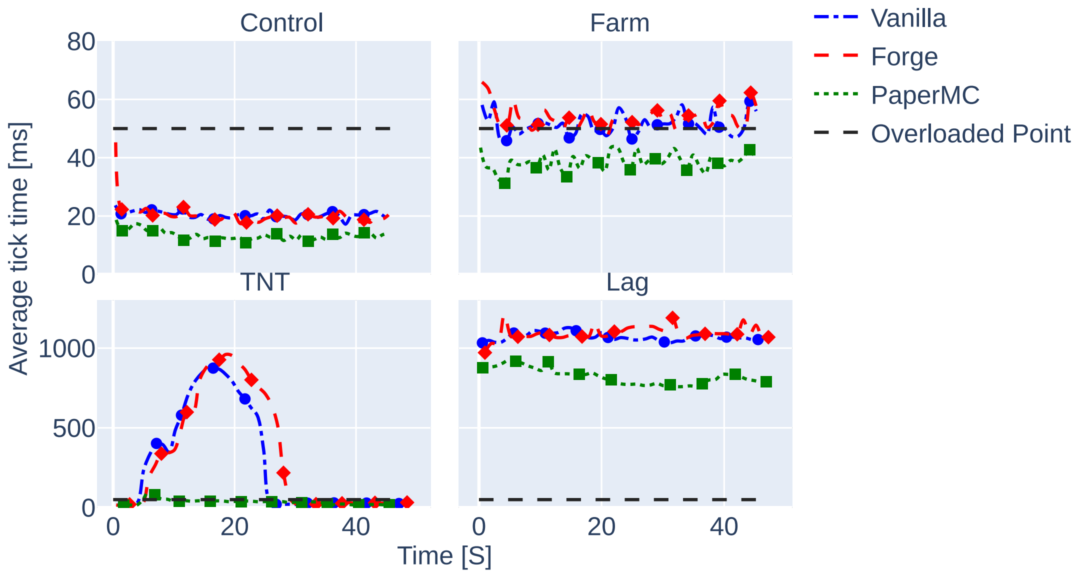

Figure 6: Tick time averages over time during experiment workloads.

Figure 6 depicts the average tick time for the Minecraft-like services Vanilla, Forge and PaperMC when running the experimental workloads. It shows that complex workloads are capable of overloading MVE services, and that of services tested PaperMC was most performant.

Using the Control workload (Figure 6, top left), none of the services become overloaded. This was expected as the Control workload contains no significant environment complexity. PaperMC stays consistently 5ms faster than the other services.

With the TNT workload (bottom left), both the Vanilla and Forge service are drastically impacted, with the Forge server reaching 1000 ms tick time, 20 times the overloaded point, and taking 24 seconds to return to a non-overloaded state. The Vanilla service peaks

Exp.	Service	Q0	Q1	Q2	Q3	Q4	Mean	Dev.
	Vanilla	17.22	19.72	20.44	21.11	23.68	20.42	1.18
Control	Forge	17.44	19.02	19.86	20.62	45.27	20.48	4.05
	PaperMC	10.87	12.33	13.06	14.15	18.67	13.42	1.63
	Vanilla	19.23	26.29	30.21	579.45	874.59	288.15	327.07
TNT	Forge	13.17	26.00	42.81	639.14	959.64	319.41	357.00
	PaperMC	14.12	19.61	35.71	40.87	106.54	34.78	17.07
	Vanilla	45.84	49.34	50.73	53.04	59.36	51.42	3.55
Farm	Forge	47.48	51.40	52.91	55.27	65.94	53.83	3.81
	PaperMC	31.23	36.61	38.54	40.44	43.67	38.58	2.93
	Vanilla	1032.94	1053.64	1066.03	1082.27	1127.49	1069.98	23.79
Lag	Forge	971.58	1072.49	1086.84	1110.11	1213.31	1094.02	42.12
	PaperMC	758.85	776.09	818.71	845.05	927.64	822.71	49.95

Table 2: Statistical summary of tick time averages over experiment workloads. Q0 - Q4 are quartiles, exp. is the experiment, dev. is standard deviation, all values are in milliseconds.

at 900 ms, and returns to nominal in 20 seconds. Despite the impact upon the Vanilla and Forge service, PaperMC becomes overloaded by only a few milliseconds, peaking at 100 ms and decreasing to normal in around a second. This is of particular interest, as Forge's custom modding API makes it the most popular and often only choice for operating modded services. An extremely popular category of mods exclusively exist to increase the scale of explosions: see for instance the ICBM mod, which has over 3 Million downloads and exclusively runs on Forge servers [8]. However, these results show that Forge performs the worst of tested services when simulating large TNT chain reactions.

The Farm workload (top right) shows a periodic fluctuation in tick time for all three services corresponding to the pulses of an in-environment player defined logic clock that controls the operation of the resource farms. This shows how directly environment and performance are correlated. Additionally, though the Farm workload contains no spontaneous chain reaction like the TNT workload, the Vanilla and Forge service are overloaded a majority of the experimental duration. PaperMC, though with a higher tick time than in the control experiment, does not become overloaded.

The Lag workload (bottom right) causes all three services to become and remain overloaded for the full duration of the experiment. Following the trend established by the other workloads, Forge is most impacted with a peak tick time of 1200 ms, Vanilla peaks at 1100 ms and PaperMC is least effected, with a peak tick time of 900 ms. With these tick times, the quality of all tested systems is degraded to the point of unresponsiveness. Unlike the TNT workload, where the PaperMC service avoided becoming extremely overloaded, the Lag workload overloaded PaperMC, though PaperMC remained a fixed percent faster than Vanilla and Forge. This is important, as it shows that the differences between PaperMC and the other services that allow it to remain largely unaffected by the TNT workload do not similarly apply to a Lag workload. This suggests that the TNT and Lag workload are inherently different and not caused by the same types of complexity.

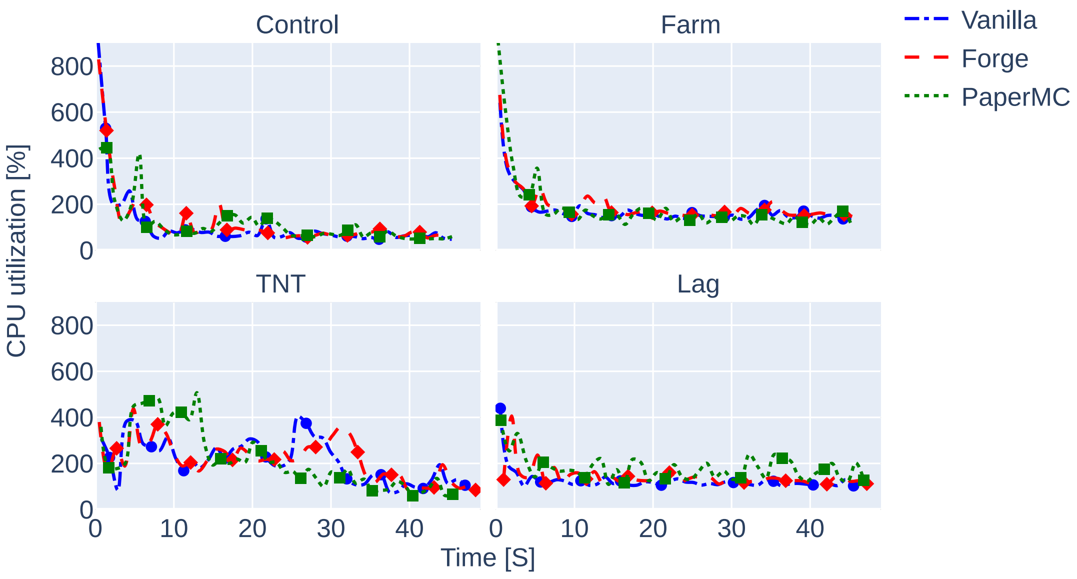

Figure 7: CPU utilization over time during experiment workloads.

Despite reacting differently, each service's performance was negatively impacted by at least one of the TNT, Farm or Lag workload. Of particular importance are the TNT and Farm experiment results, as we consider these workloads common scenarios in real world settings (Section 4), and both overloaded the Vanilla and Forge services.

5.2 MF2: Minecraft-like services are poorly parallelized.

In Figure 7 we compare CPU utilization to the tick times depicted in Figure 6. This shows that Minecraft-like services are poorly parallelized. Even during high workload periods, none of the tested services utilized more than 20% of available CPU resources.

In some workloads there is a initial spike of CPU utilization caused by the Yardstick player joining the server at the start. Aside from this, each service utilizes only a small percent of CPU resources (out of the 1600% maximum), even during periods of high tick time shown in Figure 6. We conclude that the overloaded tick time is not caused by a shortage of system resources but instead poor parallelism and contention over shared resources.

In each workload PaperMC performs better than Vanilla and Forge. This merits an explanation of the differences between PaperMC and the other services, such to drive future research into how to minimize the impact of player agnostic workloads.

The PaperMC project [44] is an open source set of community *'patches'* applied

Exp.	**Service**	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Mean**	**Dev.**
Control	Vanilla	48.00	58.93	66.20	83.20	900.43	109.30	144.48
	Forge	54.16	65.10	79.73	96.40	828.90	120.38	134.64
	PaperMC	49.20	63.03	82.53	125.53	444.90	114.41	94.70
TNT	Vanilla	74.63	129.17	214.80	274.53	403.77	213.22	91.12
	Forge	84.80	151.48	216.18	269.84	438.57	218.30	86.94
	PaperMC	59.97	122.37	177.77	282.27	507.13	216.11	131.29
Farm	Vanilla	135.50	146.30	151.80	170.27	640.73	176.07	81.92
	Forge	147.10	155.00	164.03	189.43	674.73	192.29	86.15
	PaperMC	110.97	129.97	146.13	164.43	911.27	184.59	139.66
Lag	Vanilla	100.77	107.03	116.77	123.40	439.30	125.25	50.58
	Forge	109.83	121.90	128.10	141.03	406.07	139.38	46.39
	PaperMC	109.07	137.23	166.90	205.33	387.67	178.73	54.74

Table 3: Statistical summary of CPU utilization over all experiment workloads. Q0 - Q4 are quartiles, exp. is the experiment, dev. is standard deviation, all values are in percent out of 1600.

on top of another server type (Spigot [41]) which in turn were modifications made to the Mojang-provided Vanilla server. Unlike Forge, PaperMC is not made in order to facilitate mods but instead to deal with a plethora of community grievances about server experience. As such, all PaperMC does is apply a set of source code patches and then repackage the server. The set of changes is outlined in their Github page [43], with the most important being reworking thread priorities, limited per-thread cache duplication, a custom timing scheduler, adding asynchronous loading and modification, and optimized Redstone algorithms. To show how these changes improve performance when handling world events, we compare the Flamegraph of the Vanilla and PaperMC services during the TNT workload. The Flamegraph of Forge is omitted as it is functionally identical to Vanilla's. Flamegraphs are read as follows: the y-axis shows stack depth, the width of each box shows the total (not necessarily consecutive) time a given thread was on CPU. The ordering of boxes on the x-axis is arbitrary, in this case they are in alphabetical order.

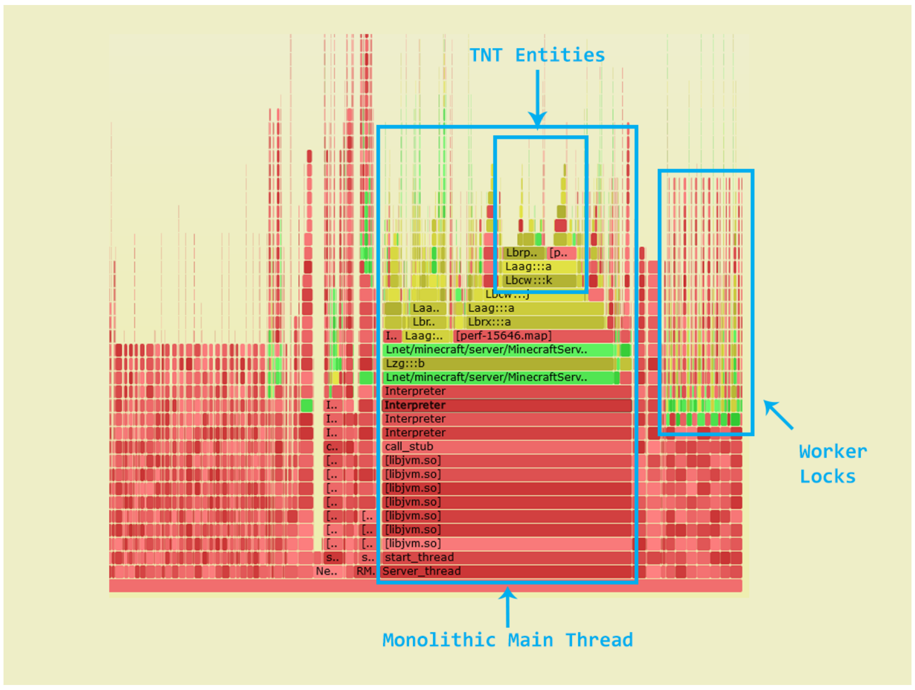

Figure 8: Vanilla flamegraph during TNT workload.

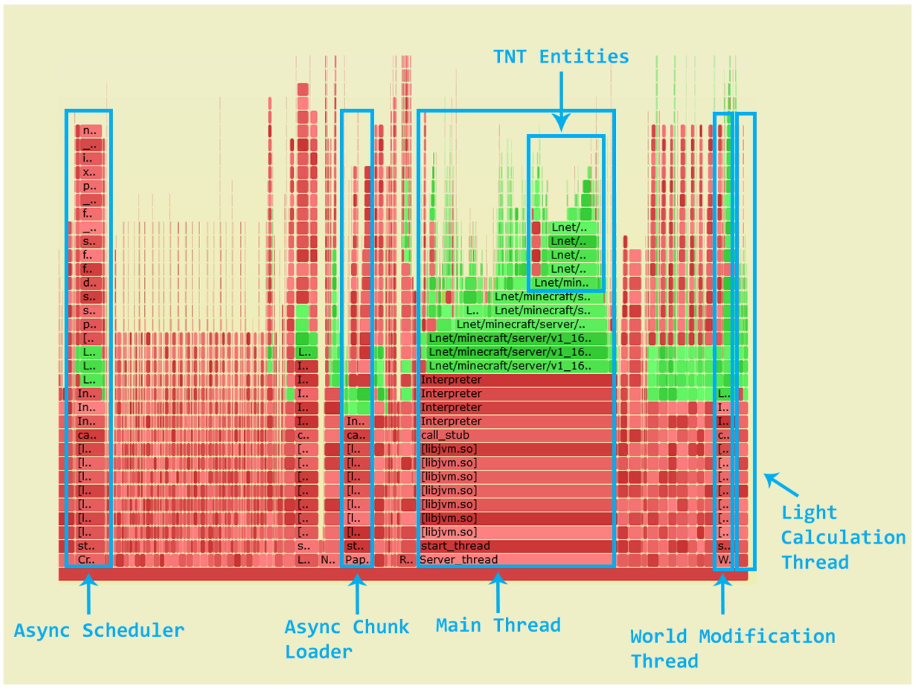

Figure 9: PaperMC flamegraph during TNT workload.

Figures 8 and 9 show that PaperMC hoists a number of environment operations out of the main thread into dedicated threads handled by a separate asynchronous scheduler thread. Thus the amount of time spent on worker locks is decreased, minimizing total resource contention. It can be extrapolated that these changes are why PaperMC handles the TNT workload better than Vanilla and Forge.

5.3 MF3: Processing the state of entities is computationally expensive.

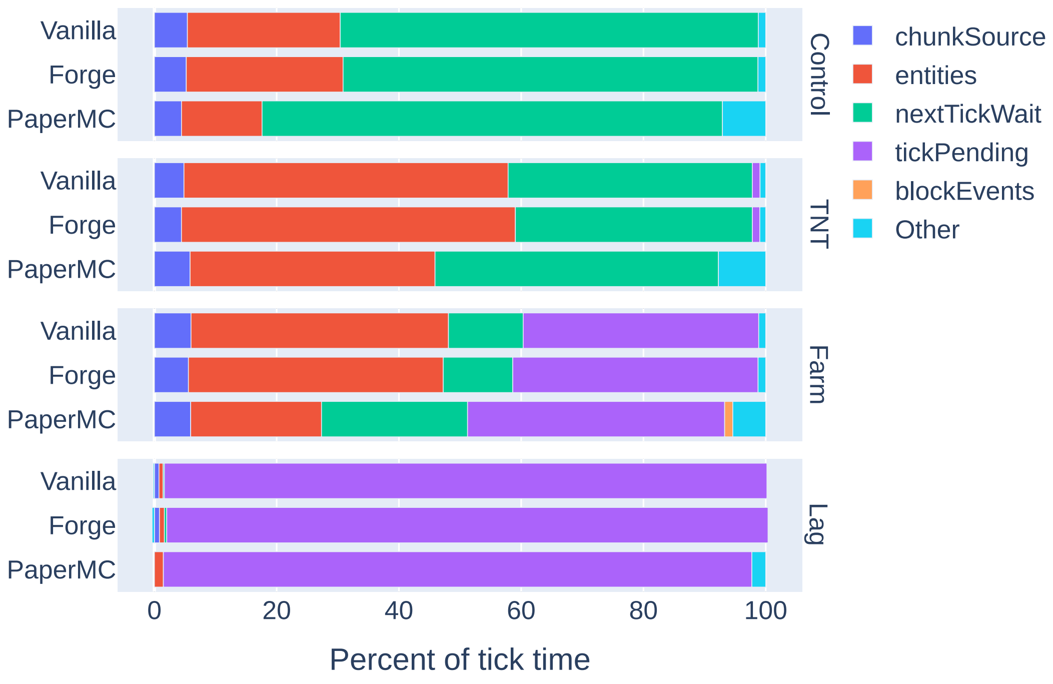

Figure 10: Tick time distribution during experiment workloads.

Figure 10 depicts the tick time distribution for the experiment split by percentage of tick time attributed to a given feature. This shows that entities account for a majority of non-waiting tick time across all workloads and services.

The values in Figure 10 are collected using the Minecraft internal server logging tool. There are two special values: "*nextTickWait*:" the percent of tick time spent idling, and "*tickPending*:" the percent of tick time spent waiting for access to shared resources. In general, a decrease in nextTickWait and an increase of tickPending means that the server is overloaded and service quality has been degraded. The other values are "*entities*," "*chunkSource*," and "*blockEvents*," referring to entity calculations, terrain modifications, and terrain calculations respectively. An "*Other*" category is included for the unspecified class and classes with values less than one percent. From these, we corroborate the statement from Section 5.1 that there is a fundamental difference between the Lag and

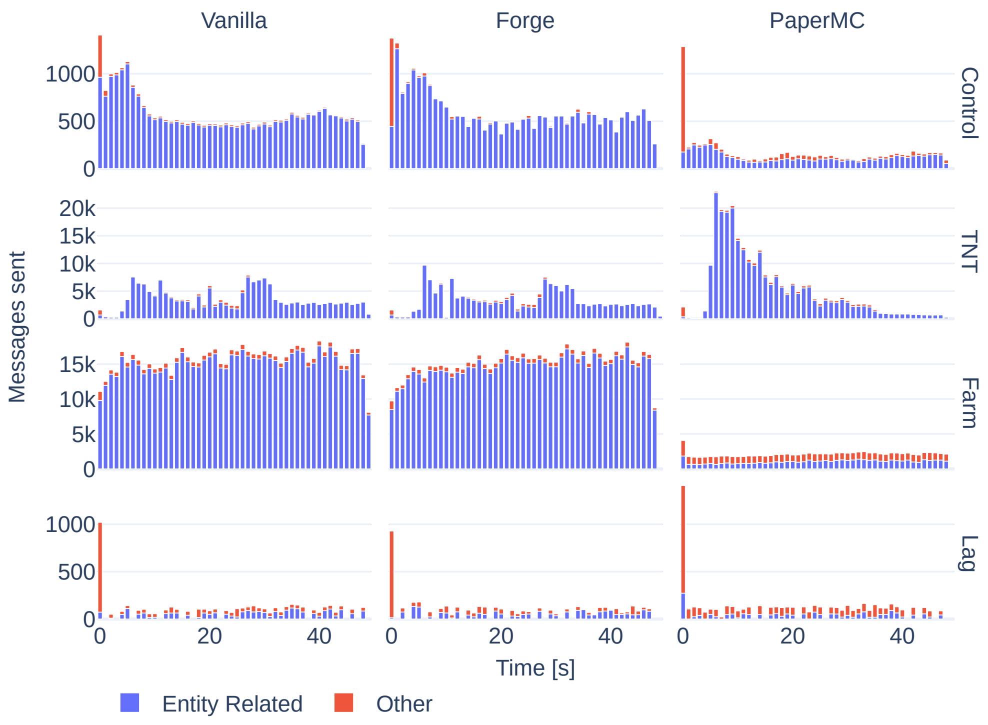

Figure 11: Messages sent over time during experiment workloads.

TNT workload, despite their similarity in overloading magnitude seen in Figure 6. The Lag workload's computational requirement comes from a large volume of shared resource accesses, whereas for TNT it is due to a high amount of entity calculations.

Figure 11 shows the messages sent by the server over the duration of the experiment, separated by whether the message relates to an entity update. For all workloads except Lag, entity related updates form more than 95% of messages sent. Figure 10 shows that the Lag workload resulted in so few messages because over 97% of the tick duration is taken up waiting on shared resources, thus the servers did not have enough time to compute and send entity updates.

Throughout all workloads and services entities account for a majority of both non-waiting tick time and server messages. When in overloaded states, MVE services send far fewer messages. This is why during the TNT workload PaperMC sent more messages than the other services despite sending less during all other workloads. PaperMC did not become highly overloaded during the TNT workload, and thus was was capable of generating and sending messages, whereas Vanilla and Forge could not.

6 Related Work

Prior research on game environment benchmarking and scalability is limited, as a majority of the work done on video game scalability is done within the field of networking optimizations to maximize concurrent player count, such as through Peer to Peer or network partitioning. Additionally, the research that does focus upon the game environment itself does not consider the unique case of Modifiable Virtual Environments. For instance, Farooq and Glauert [11] suggest partitioning world environments by load dynamically, an approach with limited applicability to MVEs, as small, largely indivisible regions of the MVE can be computationally intensive (as was the case for all experiments in section 5).

Similarly Valadares, Lopes, and Liu [49] propose further adapting partitioning by creating Microcells, arbitrary size partitions with read only borders that can be traded between instances as load requires. Such an approach does improve upon previous partition schemes but again has only limited applicability to MVEs, as a read only borders and state propagation schemes between Microcells would only increase overhead for all of the environment actions listed in section 3.

Another approach to environment scaling include interest management, as proposed by Boulanger, Kienzle, and Verbrugge [3] as well as Liu and Theodoropoulos [23]. Interest management too has flaws when applied to MVEs, as it is difficult to judge player interest in sections of the world based upon spacial or interaction features. For instance, a player can be expected to be interested in the functioning an automated farm, even when the player is not nearby.

Research that has specifically focused upon scalability in Minecraft-like games includes Van Der Sar, Donkervliet, and Iosup [50] who implement the Yardstick tool used in this work and analyse the scalability profile of common Minecraft services. However, their research does not measure the effect that Player Agnostic systems have upon scalability. Additionally, Donkervliet, Trivedi, and Iosup [9] outline a variety of methods to enhance MVE scalability, including serverless and inconsistency management, though no research has yet been done on the impacts of these techniques when used with Player Agnostic systems.

7 Discussion

In this work we first defined a classification model for Player Agnostic workloads and then analyzed their performance profile in Minecraft services. We have shown the applicability of this model to three highly popular Minecraft servers, but have not conclusively done so for other Minecraft-Like games. Additionally, we did not quantify the effect upon the experiment results from performance monitoring tool overhead or from the atypical setup of having the clients to the Minecraft service be within the same distributed cluster network.

Similarly, we have not examined the effect of player-agnostic workloads upon subjective metrics such as player experience. This is especially important in the case of PaperMC, as we have shown that it performs measurable better than the other tested services, but we have not shown that this results in better player experience. PaperMC's performance gain could come at a cost of world expressiveness, such as through limiting entity actions or world updates.

Despite these drawbacks, we have established a clear correlation between environment workload and service performance, such that the game environment alone can overload modern MVE technology.

Anecdotally, we have shown that the situation wherein the server is overloaded due to environmental calculations is common in real world systems, as we tested only workloads that are established behaviour within the Minecraft community.

8 Conclusion

Minecraft-Like services which utilize Modifiable Virtual Environments have significant societal importance for social, educational and activist efforts, and are increasingly common to operate in high performance data-centers. Prior research has shown need for better scalability in MVE technology. Towards this cause, we have defined a new class of MVE scalability problem: Player Agnostic workloads (**RQ1**). We then defined a model to represent these workloads and designed experiments (**RQ2**) to evaluate their effect on MVE performance. Thus, in answering **RQ3**, we have found that:

- **MF1:** Player-Agnostic Workloads can significantly affect the performance of Minecraft-Like games (Section 5.1). Complex sections of MVEs are capable of overloading modern MVE server technology by up to 20 times the overloaded threshold, and there are large performance differences between Minecraft-like services. Of those tested, PaperMC performed the best while Forge performed worst.
- **MF2:** The Minecraft-like services we studied, including the commercial (”vanilla”) Minecraft, are poorly parallelized (Section 5.2). This is evidenced by the trend of low CPU utilization during periods with high tick times. Even during high workload periods, none of the tested services utilized more than 20% of available CPU resources.
- **MF3:** Processing the state of entities is computationally expensive and can cause server overloads (Section 5.3). We find that 95% of all server messages are Entity related, and that 70% of non-idle tick time is Entity calculations.

We are now exploring the applicability of current and novel scalability techniques in minimizing the impact of Player Agnostic systems. Additionally, we want to evaluate more Minecraft-like games, specifically those not utilizing the Minecraft protocol.

References

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

[2] 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.

[3] J.-S. Boulanger, J. Kienzle, and C. Verbrugge. Comparing interest management algorithms for massively multiplayer games. In A. D. Cheok and Y. Ishibashi, editors, *Proceedings of the 5th Workshop on Network and System Support for Games, NETGAMES 2006, Singapore, October 30-31, 2006*, page 6. ACM, 2006.

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

[5] Circuit Network. `https://wiki.factorio.com/Circuit_network`. Accessed: 2021-01-18.

[6] M. Cocar, R. Harris, and Y. Khmelevsky. Utilizing minecraft bots to optimize game server performance and deployment. In *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–5, 2017.

[7] Conveyor Belt. `https://satisfactory.gamepedia.com/Conveyor_Belt`. Accessed: 2021-01-18.

[8] DarkGuardsman. Icbm - classic, 2020. Accessed: 2021-02-10.

[9] 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.

[10] DudeItsRocky. "huge minecraft tnt world explosion with aftermath" (minecraft tnt explosion, minecraft explosion), 2018. Accessed: 2021-02-10.

[11] U. Farooq and J. Glauert. A dynamic load distribution algorithm for virtual worlds. *J. Digit. Inf. Manag.*, 8(3):181–189, 2010.

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

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

[14] D. Gildeh. jmxquery 0.6.0, 2019. Accessed: 2021-02-10.

[15] J. S. Gilmore and H. A. Engelbrecht. A Survey of State Persistency in Peer-to-Peer Massively Multiplayer Online Games. *IEEE Trans. Parallel Distrib. Syst.*, 23(5):818–834, 2012.

[16] gnembon. Simple hostile mob farms, minecraft 1.12 - 1.16+ (fun farms ep. 15), 2017. Accessed: 2021-02-10.

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

[18] K. S. House. Space engineers.

[19] 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.

[20] Jrudolph. perf-map-agent, 2020. Accessed: 2021-02-10.

[21] M. Jumbo. Hermitcraft 7: Episode 52 - quad industrial farm, 2020. Accessed: 2021-02-10.

[22] legomasta99. [minecraft computer engineering] - quad-core redstone computer v5.0, 2018. Accessed: 2021-02-10.

[23] E. S. Liu and G. K. Theodoropoulos. Interest management for distributed virtual environments: A survey. *ACM Comput. Surv.*, 46(4):51:1—-51:42, 2014.

[24] F. D. LLC. Minecraft forge, 2021. Accessed: 2021-02-10.

[25] Logic. `https://nomanssky.gamepedia.com/Logic`. Accessed: 2021-01-18.

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

[27] Modules. `https://astroneer.gamepedia.com/Modules`. Accessed: 2021-01-18.

[28] Mojang. Minecraft. `https://www.minecraft.net/en-us/`. Accessed: 2021-01-18.

[29] Mojang. Download the minecraft: Java edition server, 2021. Accessed: 2021-02-10.

[30] Mysticat. Minecraft item sorting system: Easy & expandable tutorial 1.16, 2020. Accessed: 2021-02-10.

[31] Newzoo. Newzoo Global Games Market Report. `https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2020-light-version/`, 2020. Accessed: 2021-01-17.

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

[33] M. V. Raluca Diaconu, Joaquín Keller. Manycraft: Scaling minecraft to millions. In *NetGames, Dec 2013*, pages 1–6, 2013.

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

[35] Redstone. `https://minecraft.fandom.com/wiki/Redstone`. Accessed: 2021-01-18.

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

[37] 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.

[38] G. Rodola. psutil 5.8.0, 2020. Accessed: 2021-02-10.

[39] Shulkercraft. Minecraft fully automatic cobblestone farm - 120,000 cobble per hour - 1.16/1.15, 2020. Accessed: 2021-02-10.

[40] J. Sirani. Top 10 best-selling video games of all time, 2019. Accessed: 2021-04-20.

[41] Spigot. Spigotmc, 2020. Accessed: 2021-02-10.

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

[43] P. Team. Papermc/paper/spigot-server-patches, 2020. Accessed: 2021-02-10.

[44] T. P. Team. Papermc - the high performance fork, 2021. Accessed: 2021-02-10.

[45] D. B. Terry, V. Prabhakaran, R. Kotla, M. Balakrishnan, M. K. Aguilera, and H. Abu-Libdeh. Consistency-based service level agreements for cloud storage. In M. Kaminsky and M. Dahlin, editors, *ACM SIGOPS 24th Symposium on Operating Systems Principles, SOSP '13, Farmington, PA, USA, November 3-6, 2013*, pages 309–324. ACM, 2013.

[46] timeam. The most efficient minecraft lag machine, 2020. Accessed: 2021-02-10.

[47] 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.

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

[49] A. Valadares, C. V. Lopes, and H. Liu. Enabling fine-grained load balancing for virtual worlds with distributed simulation engines. In *Proceedings of the 2014 Winter Simulation Conference*, WSC '14, page 3459–3470. IEEE Press, 2014.

[50] 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.

[51] D. Wagner. Is minecraft turing-complete? https://gaming.stackexchange.com/a/205575, 2016. Accessed: 2021-01-18.

[52] Wire. https://terraria.gamepedia.com/Wire. Accessed: 2021-01-18.