Towards Supporting Millions of Users in Modifiable Virtual Environments by Redesigning Minecraft-Like Games as Serverless Systems

Jesse Donkervliet  Animesh Trivedi  Alexandru Iosup
VU Amsterdam  VU Amsterdam  VU Amsterdam

Abstract
How can Minecraft-like games become scalable cloud services? Hundreds of Minecraft-like games, that is, games acting as modifiable virtual environments (MVEs), are currently played by over 100 million players, but surprisingly they do not scale and are frequently not published as cloud services. We envision a new architecture for large-scale MVEs, supporting much larger numbers of concurrent users by scaling up and out using serverless technology. In our vision, developers focus on the game (business) logic, while cloud providers manage resource management and scheduling (RMS) and guarantee non-functional properties. We provide a definition for MVEs, model their services and deployments, present a vision for large-scale MVEs architected as serverless systems, and suggest concrete steps towards realizing this vision.

1 Introduction
Games provide entertainment to hundreds of million of players worldwide. In 2019, this entertainment industry generated a revenue of $152.1 billion [52], which is more than four times the global box office revenue [17] and ten times the music industry’s revenue [58]. We focus in this work on an emerging type of game, exemplified by Minecraft, which offers players a modifiable virtual environment (MVE). Minecraft is one of the most popular games of all time: it has sold more than 176 million copies [55] and has over 100 million active monthly players [25] (which, in comparison, is more than the global number of MacOS users [54]). Surprisingly, Minecraft uses replicated, non-communicating game instances, which do not scale beyond a few hundred concurrent players [68]. We envision MVEs with millions of concurrent players together in a seamless virtual world. In this paper we make a case to redesign these MVE environments as serverless systems.

The unique features of MVEs are generally beneficial. By allowing users to construct and deconstruct the world in complex ways, MVEs enable creative user-behavior that is currently not possible in other games. MVEs are also useful for other important societal tasks. Minecraft: Education Edition contains lessons on a diverse set of topics, for example, computer science lessons in which students construct their own digital computers and history lessons in which they explore UNESCO world-heritage sites [1]. Minecraft has also been used for social activism, for example, in protecting Europe’s last primeval forest from illegal logging [51].

Through their unique features, MVEs also inspire large communities of developers and players. Modifications (mods) are programmatic game changes, ranging from user-interface tweaks to entire sub-games within the MVE, designed and developed by communities and individuals outside the control of the original game developer or publisher. Modding is popular for MVEs, e.g., there are currently over 55,000 mods available for Minecraft alone [2].

We conclude there exists a strong case for supporting MVEs through a new class of cloud services, with the lucrative potential of hundreds of millions of players. However, we argue that enabling cloud services for MVEs requires a careful investigation, motivated by scalability issues.

Although Minecraft has more than 100 million of active players, there is a looming scalability challenge. Minecraft’s current scalability is only achieved by replicating game instances that do not exchange state. Thus, players join a single instance, and do not interact in-game with users on other instances. Moreover, both recent academic findings and industry practice indicate the scalability of a single game’s instance (analyzed in Section 2.1) is limited. Minecraft instances, and several community-created games using the same protocol, scale only to 300–400 players under favorable condi-
tions [68], and Minecraft Realms, a cloud-operated Minecraft service, does not allow more than 10 simultaneous players in each world-instance [48]. Many games similar to Minecraft likely suffer from similar scalability limitations.

State-of-the-art approaches for scaling traditional online-games, such as world-partitioning [26] and interest management techniques [10, 43], are designed to address scalability issues caused by player-avatars, and assume largely immutable, low-complexity virtual worlds. In contrast, MVE players have fine-grained control over the environment by modifying its terrain, components, and programs.

**Our vision**: Minecraft-like games and, more generally, MVEs, will become cloud-based services scalable to millions of concurrent players (users).

Toward our vision, and aligned with the goals and methods of the research program on massivizing computer systems [31], we propose a novel architecture where MVEs are redesigned as serverless systems, making large-scale MVEs a “killer application” for modern cloud infrastructures. In this article, we set a course towards this vision:

1. We model current MVEs as an set of interconnecting services (in Section 2). We model the services operating a current game instance, and describe how these game instances are currently deployed as cloud services.
2. We envision for large-scale MVEs (in Section 3). In our vision, MVEs are serverless; running as a collection of services. We describe for these services their operation, state synchronization, and deployment.
3. To move towards our vision, we propose three challenging and timely areas of research (in Section 4). The areas are: using serverless functions to schedule MVE services, dynamic consistency units for MVEs, and scheduling MVE services at cloudlets.

## 2 MVE Model

In this section, we model the services operating a current game instance, and describe how the scalability of these game instances is limited, despite their cloud deployment.

### 2.1 Inside MVEs: the Minecraft-like Game

This section discusses the common distributed architecture used for MVEs, depicted in the top part of Figure 1.

MVEs typically use client-server architectures, where each player runs locally a client that connects to a remote server. The client caches in memory the part of the world which is

---

1 Anecdotally, the authors of this vision paper have personally experienced the lack of scalability of several Minecraft alternatives. No systematic study of this situation currently exists.

---

Figure 1: MVE processes and services (top row), and current deployment models (bottom row).
maintaining own infrastructure. For example, many players deploy Minecraft under this model, but other Minecraft-like games can require considerable tuning and maintenance.

Following pioneering work arguing during the late-2000s for operating online games in clouds as opposed to self-hosting [40, 49], MVEs are starting to be offered as cloud-based services. There are two key models of cloud deployment. Firstly, in the Game as a Service model (D2), players pay a fee to the cloud operator, who takes responsibility for the infrastructure and resource management and scheduling (RMS) required to run the MVE server. Players run only their client software, and connect to the cloud-based MVE server. Minecraft Realms exemplifies these services.

Secondly, in the streamed games model² (D3), both the server and the client are moved to the cloud. Players run only a thin client, which sends their inputs to the cloud, and receives a fully-rendered video-stream of the player’s local view. This moves the responsibility of rendering to the cloud, giving access to the virtual world even to resource-constraint devices. However, in contrast to the previous methods, it requires high-bandwidth, low-latency, and low-jitter network connections from player to cloud datacenter [3, 13].

The three deployment models move the client and server between the user-owned and cloud-operated infrastructure, but limit individual service-scaling.

3 Vision: Serverless MVEs

Single MVE instances do not scale beyond a few hundred players [68], despite increasingly being deployed in clouds. Towards scaling to millions of single-world players, we envision serverless MVEs. In our vision, depicted in Figure 2:

1. MVEs become serverless: they are split into services, operated professionally by the cloud operator, and provisioned on-demand by the game developer or operator. This improves elasticity and multi-tenancy.
2. MVEs require specialized consistency models, which take into account the trade-off between various quality of service (QoS) (e.g., latency) and quality of experience (QoE) guarantees. This offers ways to control the bandwidth budget yet offer good gameplay experience.
3. MVE services run on the user’s device, the cloud, or on cloudlets, depending on available resources and requirements. This enables QoS and QoE improvements.

3.1 Serverless Operation

While state-of-the-art scalability techniques assume servers to primarily simulate players, MVEs can additionally contain arbitrarily complex connections of components and programs that can change the entire environment with fine-granularity.

²This modeled was mislabeled as “cloud gaming” in the early 2010s, by OnLive and similar services, as a marketing stunt.

In our vision, MVEs can scale by using serverless computing. Serverless computing is an emerging cloud architecture in which systems run as a collection of small services, or functions [22, 34]. A function contains executable code which is executed in isolation by the arrival of events [4, 53, 71].

Figure 2 depicts a high-level view of the possible services; in practice, each could be comprised of several functions. We identify the virtual world and three other classes of MVE services. The virtual world services (blue, solid outline) provide an immersive experience to players. The rendering service translates game state into video through which the player experiences the virtual world. The player simulator translates player actions (e.g., mouse clicks) into game actions (e.g., terrain modification). Because these services process directly the player’s input, their response latency needs to be of 10s of milliseconds to maintain QoE, and thus should run close to the player [12, 16, 76]. NPCs, components, and programs are simulated using independent service instances. The latter two services are unique to MVEs. To allow interaction between the services, they exchange volatile state through a system-wide state synchronization service, and persistent game state through a distributed database. Because these services can interact with all players, they run in a central cloud.

The other three important classes of MVE services (gray, dashed outline) are (i) game analytics, which enables game operators to extract information from the large amounts of data produced by the game; (ii) procedural content generation (PCG), which uses world generators to create an endless virtual world; and (iii) meta-gaming network services, which
allow players to interact outside the virtual world. Gameplay
broadcasting is a popular meta-gaming service, which allows
players to live-stream gameplay to thousands of viewers [32].
Although these services are not part of the interactive virtual
world, they provide an important supporting role and fit well
in our vision of serverless MVEs.

In our vision of serverless MVEs, its services are inde-
pendently scheduled by the cloud operator, leading to high
scalability and elasticity. How players interact with each other
and with the environment has a significant effect on the work-
load, even for traditional games [50]. The unique features of
MVEs can magnify this phenomenon [68]. Scheduling many
small, independent services is conveniently parallel, which
leads to high scalability. Moreover, adding and removing ser-
ices is fine-grained, leading to good elasticity properties [27].
(Services that need to exchange state, so where consistency is
an issue, are considered in Section 3.2.)

We conjecture serverless operation could lead to perfor-
mance isolation for MVEs, even under multi-tenancy. For
example, a serverless rendering service (as shown in Figure 2)
can be specialized to execute a single task (i.e., rendering),
thus, ensuring low performance variability in gaming [35].
This simplifies performance isolation and makes it easier for a
cloud operator to consolidate multiple such services together
on the same hardware, even in the presence of accelerated
hardware, for example, GPUs [41].

Building MVEs as a collection of services increases the
modularity of these systems, making it easier for players to
create content (e.g., mods), a driving force behind the popu-
larity of Minecraft and commercially available game engines.

3.2 Specialized Consistency Models for
An Immersive Virtual World

Essential for the MVE is to provide an immersive virtual
world where players must not experience significant inconsis-
tency with other players, when performing one of the follow-
ing four classes of in-game actions: (i) modifying regular ob-
jects, which entails adding, changing, or removing them from
the world; (ii) modifying the game-world, e.g., allowing play-
ers to construct and remove world-parts such as mountains,
valleys, oceans, and other natural phenomena; (iii) connecting
in-game components, to create new complex constructions
that may not be envisioned by the game’s developers; and (iv)
supporting player-written, arbitrary programs, as a way to
interact with the world with precise control. The last three
classes distinguish MVEs from other (popular) game-genres.

Maintaining a real-time completely consistent MVEs game
state is challenging due to large communication overheads.
We envision a system-wide service that dynamically deter-
mines the importance of each player action to all other nearby
players, and synchronizes these using a distributed state syn-
chronization service. This service uses consistency models
that quantify the amount of inconsistency in the system, and

can be specialized to execute a single task (i.e., rendering),
thus, ensuring low performance variability in gaming [35].
This simplifies performance isolation and makes it easier for a
cloud operator to consolidate multiple such services together
on the same hardware, even in the presence of accelerated
hardware, for example, GPUs [41].

Building MVEs as a collection of services increases the
modularity of these systems, making it easier for players to
create content (e.g., mods), a driving force behind the popu-
larity of Minecraft and commercially available game engines.

3.2 Specialized Consistency Models for
An Immersive Virtual World

Essential for the MVE is to provide an immersive virtual
world where players must not experience significant inconsis-
tency with other players, when performing one of the follow-
ing four classes of in-game actions: (i) modifying regular ob-
jects, which entails adding, changing, or removing them from
the world; (ii) modifying the game-world, e.g., allowing play-
ers to construct and remove world-parts such as mountains,
valleys, oceans, and other natural phenomena; (iii) connecting
in-game components, to create new complex constructions
that may not be envisioned by the game’s developers; and (iv)
supporting player-written, arbitrary programs, as a way to
interact with the world with precise control. The last three
classes distinguish MVEs from other (popular) game-genres.

Maintaining a real-time completely consistent MVEs game
state is challenging due to large communication overheads.
We envision a system-wide service that dynamically deter-
mines the importance of each player action to all other nearby
players, and synchronizes these using a distributed state syn-
chronization service. This service uses consistency models
that quantify the amount of inconsistency in the system, and

C1: Serverless functions for independent scheduling
of MVE services.

While MVE services such as world generation may be
easily expressed as serverless functions, important research
challenges remain. Providing MVE services using server-
less functions requires guaranteeing non-functional proper-

3.3 Differentiated Deployment for
Different MVE Services

Current deployment models used by MVEs, as shown in Fig-
ure 1, do not fit well the requirements for immersive virtual
worlds and other MVE services. In our vision, MVE services
are deployed across user devices, clouds, and cloudlets, de-
pending on available resources and QoS constraints.

Due to stringent QoS requirements, rendering is typically
performed on the user device. However, this can be challeng-

ging for devices with limited resources (e.g., smartphones).
Scheduling these services on cloudlets provides a comprome-

3 Towards Scalable, Cloud-Based MVEs

Matching the vision described in Section 3, we describe here
three timely research challenges towards supporting millions
of players in serverless MVEs. We aim to design novel solu-
tions to these challenges in future work, and apply them to an
open-source MVE. To evaluate their scalability improvements,
we aim to observe application-level metrics (e.g., maximum
number of players, simulation rate) and system-level metrics
(e.g., instructions per cycle, bandwidth usage) while bench-
marking the system using realistic and heavy workloads (e.g.,
many players that build and connect components).
ties, which is an open research problem [22]. More advanced features of serverless computing, such as keeping state and communicating with other functions using networking [65] or shared storage [36, 37, 56] are actively areas of research, but may be required to support more complex MVE services. Combining these features may enable a serverless version of distributed actor models [5] (or Actors as a Service [63]), which matches well our proposed architecture.

Using serverless architectures is already being explored for MVE-related services. For example, PyWren [33] and Graphless [67] are serverless systems for data- and graph-processing, respectively, and may enable game analytics.

C2: Dynamic consistency units to control and limit MVE inconsistency.

Continuous consistency [74] is a consistency model that quantifies the inconsistency in a system, and allows the system to operate under arbitrary consistency bounds.

Unfortunately, the real-time nature of MVEs does not tolerate unavailability—some value must always be returned to (rendered for) the user. To address this problem, we propose dynamic consistency units as a timely research topic. In such a model, MVE operators specify policies that dynamically determine inconsistency bounds for each player, and even for each type of data—world objects, terrain, etc [66]. A dedicated consistency service then communicates updates to MVE services based on their bounds. We aim to evaluate experimentally the potential for scalability of these models.

C3: Schedule services at cloudlets to resources to improve QoS and cost.

Cloudlets are resource-rich compute clusters connected to the Internet that are specifically available for nearby devices [62], which play an important role in edge computing, where such resources are deployed near the Internet’s edge, close to end users [61]. Examples include servers available at internet service provider (ISP) end-stations and cell-towers.

Initial research on using cloudlets for gaming is promising. Computational offloading to cloudlets [20] enables high QoS for players with resource constrained devices. Streaming games from cloudlets can deliver low-latency video to users, while reducing bandwidth load from remote clouds [42, 76]. Game analytics may run on processing engines specifically designed for cloudlets at the Internet’s edge [24]. Further research could investigate if rendering can take place at multiple cloudlet locations concurrently, to efficiently provide gameplay broadcasting for live audiences. Evaluating the benefits of running other computational MVE services on cloudlets, such as world generation, remains an open research challenge.

The community is exploring how to move serverless systems to the edge [57], and how to use efficiently cloudlets for game workloads [15]. For example, resource sharing between small units of computation on cloudlets [59] is particularly relevant to serverless MVEs. Data storage [72] or caching [75] could make data from all four game functions available to users with low latency and reduced cost. Further research should explore using physical hot-spots as gaming caches.

5 Related Work

We propose here a novel, integrated vision of scalable MVEs, which we see achieved through a combination of diverse systems-level techniques and cloud deployment models.

This article is not the first that argues for increasing the scalability of MVEs. Manycraft [19] increases the scalability of single Minecraft instances to support up to 1,000 players in a static world. Koekepan [21] modifies Minecraft to be used as a research platform and adds scalability through world partitioning. The Yardstick [68] benchmark was used to show Minecraft and other MVEs do not scale well.

The MVE services identified in this article can be built using distributed actors. Implementations of this model are available (e.g., Akka and Orleans [7, 11]), but do not include specialized consistency models for latency-sensitive MVE state changes, and require RMS from the game operator.

The approach we propose leverages existing techniques for scaling horizontally and vertically across servers. Common approaches include partitioning geographical areas of the virtual world [18, 30, 39, 45, 70], and partitioning based on entities [14, 28, 29, 46]. Our vision is a generalization of these approaches, allowing independent scheduling.

Last, dynamic consistency is different from scalability techniques used in traditional large-scale online games [73], such as consistency models for areas of interest [10, 43, 69] and reduced synchronization [8, 9, 47, 60, 64], and consistency models for key-value stores, e.g., Cassandra [38], Pileus [66].

6 Conclusion

In this work, we focus on an emerging type of game, exemplified by Minecraft, which offers players a modifiable virtual environment (MVE). Although Minecraft is one of the most popular games of all time, it uses replicated, non-communicating game instances, which do not scale beyond a few hundreds of concurrent players.

We propose large-scale MVEs, run as a collection cloud-native services, as a “killer application” for modern cloud infrastructures. In this article, we set a course towards this vision with a three-fold contribution. Firstly, we model current MVEs as a system of interconnecting services, and describe how these game instances are currently running as cloud services. Secondly, we envision serverless MVEs, which run as independently scheduled services that synchronize using specialized consistency models. Thirdly, we propose serverless computing, dynamic consistency units, and scheduling MVE services on cloudlets as challenging and timely areas of research to move towards this vision.
7 Discussion Topics

Discussion Point and Feedback

1. How to deal with technical limitations of serverless platforms, such as bounded execution time and limited communication possibilities?
2. How to leverage existing consistency models and performance techniques from the area of datacenter storage? We understand that there is a large body of work in the area [38,66] which are optimized for key-value stores. Can we adapt these models for gaming workloads (many writes) and requirements (real-time)?
3. On the use of specialized consistency models and protocols in clouds. We would like a discussion around the topic, not limited to our challenge C2. Are there also practical issues to be considered? Is there experience with running such services in clouds, especially related to performance variability in networks?
4. On the use of cloud-edge resources. How does the community see the practical aspects of such approaches? Any use-cases related to ours?

Controversial Topics

1. Are gaming and MVEs important topics to the HotCloud community? The current paper discusses challenges specifically in the context of gaming, which by itself affects many people, but also represents a broader class of emerging AR/VR applications that interact with a shared world. All these applications need predictable low-latency decision making and data processing, access to a shared state, and flexible scalability. We believe starting with gaming may synthesize reusable abstractions, mechanisms, and ideas applicable to other domains.
2. Are serverless architectures a good fit for MVEs? Players’ ability to add and remove (often independent) simulated components to the virtual world fits well with the elasticity provided by serverless architectures. However, MVE simulations may share state with each other, and may need to run for longer periods of time. We are familiar with the many platforms used in serverless operations in the cloud, related analysis studies [6,22,34,44], and use-cases [23,33], but know that one size does not fit all, and we would appreciate feedback and a discussion on the experience of the community in running serverless workloads, especially large-scale and/or with interactive elements.

Acknowledgments

We thank Aurojit Panda and the anonymous reviewers. Work partially funded by the Dutch NWO project MagnaData.

References


